ESPORA: Definición de Lenguajes de Operación Específicos de Dominio Siguiendo un Proceso de Desarrollo Dirigido por Modelos

David Musat Salvador

Jennifer Pérez Benedí

Pedro P. Alarcón Cavero

GRUPO DE INVESTIGACIÓN SYST

(http://syst.eui.upm.es)

UNIVERSIDAD POLITÉCNICA DE MADRID

Febrero de 2009

Agradecimientos

Este trabajo ha sido financiado por la Universidad Politécnica de Madrid y la Comunidad de Madrid. Proyecto de creación y consolidación de Grupos de Investigación de la UPM, Modalidad B, Entorno Reconfigurable para la validación de sistemas autónomos. RETVAS: Reconfigurable Testing Environment for Validation of Autonomic Systems CCG07-UPM/TIC-1438.

1. INTRODUCCIÓN

Esta monografía se enmarca dentro del programa de creación y consolidación de Grupos de Investigación de la UPM, Modalidad B, concretamente en el proyecto CCG07-UPM/TIC-1438 "Entorno Reconfigurable para la validación de sistemas autónomos" de nombre RETVAS: Reconfigurable Testing Environment for Validation of Autonomic Systems. Como actividad propuesta en el desarrollo de dicho proyecto se incluyó la realización de una serie de proyectos fin de carrera por parte de estudiantes de últimos cursos de las titulaciones de Ingeniero Técnico en Informática de Sistemas e Ingeniero Técnico en Informática de Gestión, ambas pertenecientes a la UPM. Siguiendo la línea marcada, el presente documento contiene parte de los resultados obtenidos en el desarrollo del Proyecto Fin de Carrera por parte alumno, becado en el proyecto RETVAS, D. David Musat Salvador, estudiante de la Escuela Universitaria de Informática de la Universidad Politécnica de Madrid. Dicho Proyecto Fin de Carrera está siendo codirigido por los profesores Dra. Jennifer Pérez Benedí y Dr. Pedro P. Alarcón Cavero, ambos integrantes del grupo de investigación Syst de la UPM, y siendo investigadora principal e investigador participante respectivamente del proyecto RETVAS.

El objetivo del trabajo que aquí se presenta es el de generar una herramienta mediante un proceso de desarrollo software dirigido por modelos (MDD: Model Driven Development) que facilite la creación de lenguajes específicos de dominio orientados a la operación de sistemas intensivos software en general, y de sistemas autónomos en particular. La definición de este tipo de lenguajes se realizará de forma completamente gráfica e intuitiva por parte de los usuarios (operadores) que podrán tener un buen conocimiento del dominio específico objeto de aplicación, y dónde no sea necesario dominar técnicas ni herramientas software para la definición de lenguajes de este tipo. Los lenguajes de operación específicos de dominio obtenidos permitirán la operación, monitorización y pruebas de sistemas con software. Y

constituirán la entrada necesaria para generar dinámicamente un entorno de operación y pruebas, basado en el concepto TOPEN, desarrollado en el grupo de investigación Syst de la UPM (http://syst.eui.upm.es).

Este documento constituye una primera versión que refleja algunos de los resultados obtenidos hasta el momento en el desarrollo de un entorno reconfigurable orientado a la operación, monitorización y validación de sistemas autónomos. El documento se irá actualizando en futuras versiones con avances que se vayan produciendo en esta línea de investigación.

El contenido de este documento se ha estructurado del siguiente modo:

- En el punto 2, se introducen los conceptos fundamentales en los que se apoya el Desarrollo Dirigido por Modelos.
- En el punto 3 se describen brevemente algunas de las herramientas de gestión de modelos, y se profundiza en GMF (Graphical Modeling Framework) para Eclipse.
- En el punto 4 se describe el metamodelo de operaciones del sistema en el que se basa la herramienta de definición de lenguajes de operación específicos de dominio, esto es, la herramienta ESPORA.
- El punto 5 se detalla tanto el proceso de desarrollo de la herramienta ESPORA, como de su utilización y funcionamiento.
- En el punto 6, se aporta un caso de estudio basado en una planta de producción de biogás, que permite utilizar ESPORA como herramienta para facilitar la definición del lenguaje de operaciones de plantas de biogás.

2. DESARROLLO SOFTWARE DIRIGIDO POR MODELOS Y GESTIÓN DE MODELOS

Durante las últimas cinco décadas, los investigadores y desarrolladores de software han estado creando abstracciones que les ayuden a programar en términos de diseño y que les encapsulen de las complejidades que atañen a los entornos hardware –por ejemplo, la CPU, la memoria, y los dispositivos de red.

Desde los primeros momentos de la informática, estas abstracciones incluían tanto el lenguaje como las plataformas tecnológicas. Por ejemplo, los primeros lenguajes de programación, como los ensambladores y Fortran, abstraían a sus desarrolladores de las complejidades de programar con lenguaje máquina.

Aunque estos primeros lenguajes y plataformas aumentaron su nivel de abstracción, tenían todavía un enfoque "orientado al hardware". En particular, ellos ofrecían abstracciones del espacio solución –esto es, el dominio de las propias tecnologías del ordenador- en vez de abstracciones del espacio problema que expresa conceptos aplicados a dominio, como las telecomunicaciones. Esto se solucionó más tarde con un mayor nivel de abstracción mediante el paradigma de la ingeniería orientada a objetos y la programación estructurada. Muchos esfuerzos pasados han creado tecnologías que han aumentado el nivel de abstracción utilizado para desarrollar software.

En los años ochenta se comenzaron a utilizar las herramientas CASE (Computer-Aided Software Engineering), que estaban enfocadas a desarrollar métodos y herramientas software que permitieran a los ingenieros del software expresar sus diseños en representaciones gráficas tales como máquinas de estado, diagramas de estructura, diagramas de flujo de datos, etc. Un objetivo de las herramientas CASE era proporcionar más entornos gráficos que incurrieran en menos complejidad que los lenguajes de programación ya que se conseguía un análisis mucho

más intuitivo. Otra meta era la generación automática de código ejecutable, para así reducir el esfuerzo manual de depurar el código fuente. Finalmente, también se perseguía que los programas proporcionasen portabilidad.

Aunque estas herramientas atrajeron una considerable atención en la investigación, no se adoptaron en la industria. Un problema que presentaban era que las representaciones en el lenguaje gráfico de los programas de las herramientas CASE no eran suficientemente expresivos para tener en cuenta las necesidades de las plataformas existentes, que eran sistemas operativos como MS-DOS, OS/2, o Windows. Dichas plataformas requerían soporte para características tan importantes como la transparencia de distribución, la tolerancia a fallos o la seguridad. Además, las herramientas CASE incluían entornos de ejecución propios, lo cual hacía mucho más difícil integrar el código que generaban con otros lenguajes software y plataformas tecnológicas.

Otro problema era el hecho de no soportar el desarrollo software colaborativo, así que las herramientas CASE estaban limitadas a programas escritos por una persona o un equipo que serializase su acceso a los ficheros gestionados por la herramienta. Además no eran muy versátiles hablando en términos de usabilidad, es decir, eran demasiado genéricas y muy poco maleables a la hora de hacerlas específicas. Estos problemas junto con la cantidad y complejidad del código generado, hizo complicado el desarrollo, la depuración y evolución de las herramientas CASE y de las aplicaciones creadas con ellas. Como resultado, tuvieron muy poco impacto comercial en el desarrollo software entre los ochenta y los noventa.

Por otra parte, los avances en lenguajes y plataformas durante las dos últimas décadas han conseguido un mayor nivel de abstracción. De esta forma, el desarrollo de aplicaciones resulta mucho más cómodo para los desarrolladores, reduciendo así uno de los principales problemas de las herramientas CASE. Por ejemplo, los programadores hoy en día utilizan lenguajes

orientados a objetos, tales como C++, Java, o C# en vez de Fortran o C. Este tipo de lenguajes, que están basados en el paradigma de la orientación a objetos, consiguen una mayor abstracción siendo más cercanos al mundo real, más legibles y más entendibles debido a que utilizan el concepto de "objeto", un elemento con propiedades que representa a los que nos rodean. Como causa de la madurez de la tercera generación de lenguajes y plataformas reutilizables, la mayoría de desarrolladores están capacitados para abstraerse de las complejidades asociadas a la creación de aplicaciones utilizando tecnología punta.

Hoy en día existen nuevos problemas que atañen al desarrollo software y que aumentan su complejidad, principalmente relacionados con requisitos no funcionales. La entrada de Internet como medio de trabajo y el hecho de que la mayoría de herramientas que se usan a diario precisan de conexión a Internet ha hecho que requisitos no funcionales como la distribución, la seguridad o privacidad cobren una gran relevancia. Con esto se observa que una vez cumplidos los requisitos funcionales, hay que ir más allá y, por tanto, se deben de tener en cuenta factores como el hecho de que no puede ser lo mismo el nivel de usabilidad de una herramienta para un niño que para un adulto. Por otro lado, es obligado mencionar que con el auge de la informática existe un número elevado de empresas dedicadas al desarrollo software, y por esta razón hay una gran competitividad en el mercado. De ahí que se marque la diferencia tratando de realizar un producto que proporcione calidad. Como solución a todo esto, se tiende a realizar unos planos o guías, denominados modelos, que ayuden a que todos los ingenieros del software vayan en una misma dirección. Es en este contexto en el que aparece el propósito de desarrollar tecnologías en base a la Ingeniería Dirigida por Modelos (Mode-Driven Engineering, MDE[Fon04]).

2.1 Ingeniería Dirigida por Modelos (MDE)

La premisa principal de MDE (Model Driven Engineering) es que los programas se generarán de forma automatizada a partir de modelos. Su objetivo es superar los problemas que tenían las herramientas CASE y mejorarlas.

MDE aumenta la variedad de modos de representación de modelos (ontologías, UML[UML08], esquemas relacionales, esquemas XML [XML08] , etc). El uso de modelos para el desarrollo del software nos proporciona soluciones que son independientes de las tecnologías, y cuyo código fuente puede haber sido obtenido mediante la generación automática de código.

2.2 Desarrollo Dirigido por Modelos (MDD)

Principalmente, existen dos enfoques que aplican el paradigma MDD **¡Error! No se encuentra el origen de la referencia.** (*Model Driven Development*), MDA[MDA08] *(Model Driven Architechture)* y MDE*(Model Driven Engineering)*. MDD es un paradigma de desarrollo software que está basado en modelos que utilizan técnicas de generación automática para obtener el producto software. En MDD se incluye MDE y dentro de MDD podemos distinguir dos vertientes, "Model Driven Architecture" (MDA), propuesto por la OMG, y "Software factories"**¡Error! No se encuentra el origen de la referencia.**, propuesto por Microsoft. MDA consigue abstraer el desarrollo software, de forma que un sistema software pueda adaptarse a diferentes tecnologías y lenguajes de programación. En MDA se define modelo como "la descripción o especificación de un sistema y su ambiente para cierto propósito y frecuentemente representado como una combinación de dibujos y texto". El texto puede ser o bien un lenguaje de modelado o un lenguaje natural. Los modelos son considerados como entidades de primera clase.

"Software Factories" trata de potenciar la reutilización las arquitecturas, componentes software, técnicas y herramientas para el desarrollo software. Las "Software Factories" se desarrollan utilizando un conjunto de herramientas llamadas DSL Tools (Domain Specific Languages Tools) integradas dentro de Visual Studio 2005 y futuras versiones.

En este trabajo de fin de carrera vamos a realizar un desarrollo dirigido por modelos basándonos en MDD De esta manera, el resultado de este proyecto es genérico para ambos enfoques. Además, dicho trabajo se realizará utilizando herramientas de software libre con el objetivo de que este al alcance de todo el mundo y pueda ser utilizado por cualquier potencial usuario.

MDD aplica lecciones aprendidas de los esfuerzos realizados con las herramientas CASE para desarrollar plataformas de más alto nivel y abstracciones del lenguaje. Otra de las ventajas es la posible existencia de elementos gráficos para facilitar el modelado. Tener elementos gráficos que ayuden al ingeniero del software a realizar su cometido es muy importante ya que ayuda a la abstracción y además es muy intuitivo y visual.

MDD no impone restricciones de dominio específico y proporciona más facilidades a cuestiones de validación y pruebas, es decir aumenta la productividad, lo que es muy importante ya que no se invertiría tanto tiempo y esfuerzo en el ciclo codificación-pruebas-entrega (ver Figura 1). En la Figura 1 se puede observar el esquema teórico que se debe realizar en un proceso de desarrollo software en comparación con el

proceso que siguen muchos de los desarrolladores software. Normalmente es más fácil desarrollar y depurar herramientas MDD y aplicaciones creadas con estas herramientas.

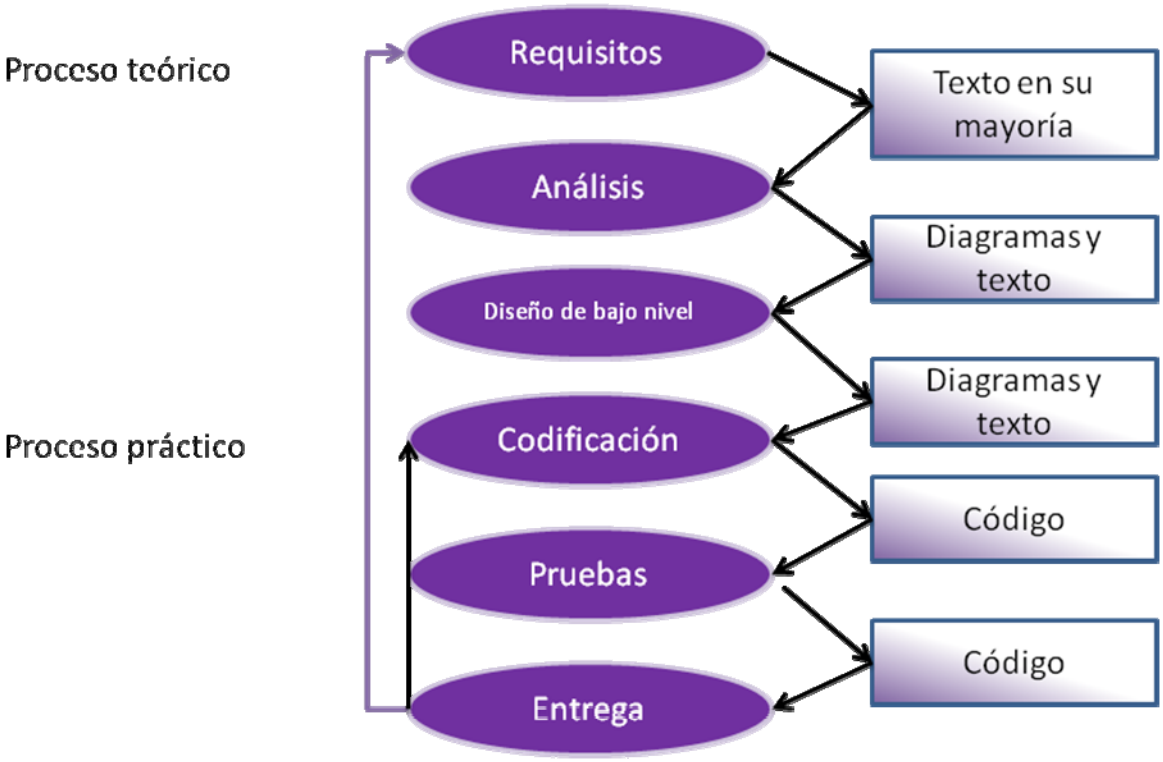


Figura 1: Proceso iterativo del desarrollo software

2.2.1 Relaciones entre modelos y metamodelos

Los metamodelos definen modelos y establecen sus propiedades de forma precisa, es decir, un metamodelo es un modelo de modelos.. Además, facilitan el mantenimiento y la automatización del desarrollo software gracias al soporte que las herramientas de modelado ofrecen actualmente.

Cuando un modelo es construido según un metamodelo bien definido, se dice que el modelo es conforme a ese metamodelo. Por ejemplo, si tenemos un documento XML, se puede validar respecto a un esquema XML y saber si satisface la relación *conforme a*. En este caso hay una clara relación entre el documento XML y el lenguaje de modelado determinado por el esquema XML. En cambio si tenemos un modelo y un sistema que lleva a la realidad tal modelo, se dice que el sistema es una *instancia* del modelo.

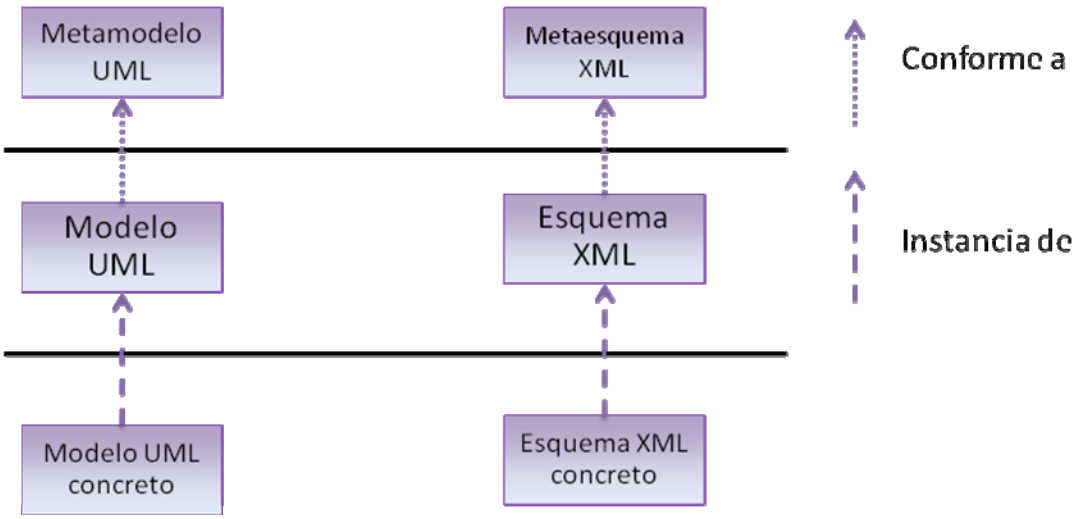


Figura 2: Relaciones *Instancia de* y *Conforme a*

El metamodelado puede aplicarse de forma sucesiva; el resultado de esto es una jerarquía de modelos que se conoce como *arquitectura de metamodelos* .Cada nivel de la pila es un nivel de modelado que permite construir los modelos del nivel inferior. En la Figura 2 se puede observar con ejemplos de XML y UML las relaciones *conforme a instancia de* que existen entre los diferentes modelos de ambos lenguajes de modelado.

Cualquier arquitectura de metamodelado presenta el problema de cerrar el nivel superior de la pila, de tal forma que no se necesite un metamodelado de nivel superior siempre. Para cerrar este nivel superior de una arquitectura de metamodelado se requiere de un modelo que sea *conforme a* sí mismo en el nivel superior de la torre, esta relación se denomina relación reflexiva *conforme a*.

2.2.2 Transformación de modelos

El proceso automatizado de utilizar uno o varios modelos como entrada y producir uno o más modelos como salida, siguiendo un conjunto de reglas de transformación se conoce como transformación de modelos. Se define transformación como la generación automática de un modelo destino a partir de un modelo origen de acuerdo a una especificación. La transformación de modelos puede realizarse de tres formas: mediante la manipulación directa del modelo a través de una API, a través de una representación intermedia que permita a otras herramientas manipular el modelo, o mediante un lenguaje de transformación de modelos, que tiene construcciones específicas para ello.

Las transformaciones de modelos pueden especificarse usando diferentes lenguajes. Sin embargo, la tendencia en MDD es usar aquellos dedicados específicamente a la transformación de modelos. La interoperación de transformaciones de modelos es la posibilidad de usar una transformación en distintas plataformas, lo cual es factible si se usan estándares como QVT (Query View Transformation)[QVT].

El proceso de transformación de modelos se efectúa a través de una función de correspondencia o transformación, compuesta por un conjunto de reglas de transformación.

La entrada de la función de transformación es un conjunto de modelos y sus metamodelos corresponden al dominio de la función. La salida es un conjunto de modelos que pertenecen a un conjunto de metamodelos de salida o codominio.

La ejecución de la función define las correspondencias abstractas entre elementos de los metamodelos de entrada y los de salida. Se define correspondencia entre modelos como la ejecución de una función de transformación, es decir, la relación entre los elementos concretos de un modelo de entrada y otro de salida, el cual no se conoce hasta que se ejecuta la transformación.

La transformación es horizontal en MDD si el modelo origen y el modelo destino pertenecen a la misma categoría de acuerdo a la jerarquía modelos o arquitectura de modelos (ver sección 2.2.1), sino es vertical.

Las transformaciones de modelos se ejecutan en una plataforma de transformación, que se compone de uno o más metamodelos para definir los metamodelos de entrada y salida de las funciones de transformación, un lenguaje de transformación y un entorno de ejecución.

2.2.3 Arquitectura Dirigida por Modelos (Model Driven Architecture (MDA))

La Arquitectura Dirigida por Modelos ("*Model Driven Architecture* o MDA") es un acercamiento al diseño de software propuesto por el "*Object Management Group* (OMG)" en el año 2001. MDA es un marco de trabajo cuyo objetivo central es resolver el problema de que el cambio de tecnología de un sistema software no sea cotoso, así como su integración en la plataforma que corresponda. La idea principal de MDA es usar modelos, de modo que las propiedades y

características de los sistemas queden plasmadas de forma abstracta, y por tanto, los modelos no se vean afectados por los cambios tecnológicos.

2.2.3.1 Conceptos básicos de MDA

MDA se ha concebido para dar soporte a la ingeniería dirigida a modelos de los sistemas software. MDA proporciona una serie de guías o patrones expresadas como modelos. MDA propone cuatro niveles de abstracción que componen la jerarquía o arquitectura de modelos (ver sección 2.2.1.). Estos son CIM (Computation Independent Model), PIM (Platform-Independent Model), PSM (Platform-Specific Model), y la aplicación final. Algunas definiciones que se han de tener en cuenta para comprender este enfoque son las siguientes:

Plataforma: Una plataforma es un conjunto de subsistemas y tecnologías que provee un conjunto de funcionalidades a través de interfaces y unos patrones específicos de uso, los cuales pueden ser empleados por cualquier aplicación sin que ésta tenga conocimientos de los detalles de cómo esta funcionalidad es implementada.

CIM: El primer modelo de la jerarquía MDA siguiendo un criterio de más abstracto a menos abstracto (ver Figura 3) es un modelo CIM. Éste modelo es una descripción de la lógica del negocio desde una perspectiva independiente de la computación. Es un modelo del dominio.

PIM: El segundo modelo que define MDA en su jerarquía en un modelo PIM. Éste es un modelo con un alto nivel de abstracción, una vista de un sistema desde el punto de vista independiente de la tecnología pero teniendo en cuenta su computación, ya que se está pensando en el sistema software y no en el conocimiento del dominio, como en un modelo CIM. Siguiendo la arquitectura MDA, un modelo PIM es posible construirlo a partir de un CIM, aunque no es estrictamente necesario.

PSM: El tercer modelo que ocupa la arquitectura MDA, es un modelo PSM. Éste es un modelo específico de plataforma, concretamente de la plataforma tecnológica donde se ejecutará el sistema. Un modelo PSM se construye a partir de un PIM, es decir, un modelo PIM puede transformase en uno o más PSM. Un PSM se encarga de especificar un sistema en términos de la plataforma en la que vaya a ejecutarse.

Aplicación Final: El último modelo en la jerarquía MDA es la aplicación final y por tanto, el paso final en el desarrollo. La aplicación final se obtiene a partir de la trasformación de un PSM a código, es decir, a un lenguaje de programación.

La siguiente figura muestra los cuatro niveles de abstracción MDA. Dentro de esta jerarquía es posible establecer, tanto jerarquías verticales como horizontales. Las transformaciones horizontales son por ejemplo de PIM a PIM o de PSM a PSM, y las vertical son por ejemplo de PIM a PSM o de PSM a código.

Figura 3: Niveles de abstracción en MDA

MDA define los modelos CIM, PIM, PSM y la aplicación final, así como la relación que debe haber entre ellos. El desarrollo de un sistema de acuerdo al marco de trabajo de MDA se inicia con la construcción de un CIM, el CIM se transforma en un PIM. Después el PIM es transformado en uno o más PSM. Por último, el código es generado a partir de los PSMs. El

patrón de transformación de MDA se muestra en la figura 3, que representa una transformación de modelos la cual requiere como entrada un CIM, y tiene como salida PSM. La transformación, además, recibe información adicional del ingeniero basada en el modelo, que sirve para indicar qué reglas aplicar, cómo transformar ciertos elemento del modelo origen en elementos del modelo destino e indicar decisiones de diseño.

Los PIM y PSM se marcan antes de la transformación, idealmente éstos no deberían contaminarse con la información que se recibe, por eso el patrón recibe el modelo como entrada (Ver Figura 4). Sin embargo, lo más común es utilizar *tagged values,* por lo que el patrón de transformación real es de PIM-marcado a PSM.

MDA dice que los modelos PIM y PSM se expresen como modelos UML y que las transformaciones deben automatizarse al máximo posible. Hay alternativas como la transformación directa, (vía programación), el uso de patrones y el metamodelado.

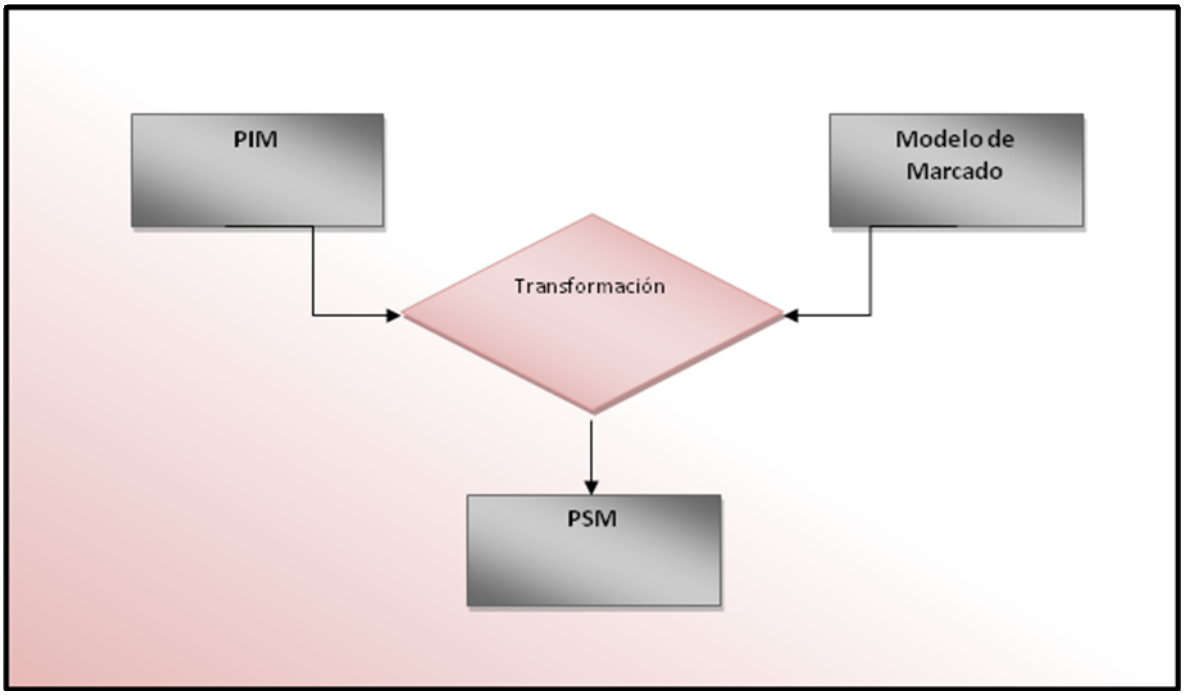


Figura 4: Patrón de transformación

Las versiones iniciales de MDA sirvieron como base de MDD, que generalizó MDA, y que define las transformaciones en el contexto del metamodelado. De ahí que en este proyecto final de carrera, se utilice MDD en lugar de MDA.

En MDD la forma más aceptada de definir modelos es a través del metamodelado, y las transformaciones a través de lenguajes de transformación de modelos. En cambio, en la propuesta original de MDA el metamodelado no era una condición necesaria. Las versiones posteriores de MDA incorporaron después ideas de MDD, que dieron lugar a *Meta-Object facility,* MOF **¡Error! No se encuentra el origen de la referencia.** y *Query Views Transformations* (QVT).

MDA es una materialización de MDD basada en las tecnologías y estándares de OMG. El estándar para el metamodelado es MOF y el lenguaje de transformación QVT.

2.2.3.2 Automatización de los pasos de transformación

Las transformaciones en MDA son siempre ejecutadas por herramientas. Muchas herramientas son capaces de transformar PSM a código sin seguir el enfoque MDA. Dado el hecho de que un PSM es muy cercano al código. Sin embargo, las transformaciones de CIM a PIM, y de PIM a PSM, resultan más complejas dentro del proceso de desarrollo software. En MDA estas transformaciones también se realizan de forma automática y aquí es donde destacan los beneficios de MDA. Esto es debido a que en esta etapa de desarrollo se invierte una gran cantidad de tiempo. Ahora, gracias a las herramientas que soportan MDA, como por ejemplo GMF, EMF y GEF, esta parte se realiza más fácilmente y se mejora el tiempo de desarrollo. Sin embargo, es de destacar que no todas las herramientas son lo suficientemente sofisticadas como para proveer de transformaciones de PIM a PSM y de PSM a código fiables al cien por cien. Pero existen herramientas actualmente que son capaces de generar una aplicación

ejecutable a partir de un PIM que proporciona una funcionalidad básica, como crear y cambiar objetos en el sistema.

2.2.3.3 Beneficios de MDA

Para aumentar la productividad y optimizar el tiempo de desarrollo, en MDA, el desarrollador debe centrarse en el PIM. Los PIM pueden provenir de un CIM y los PSM que se necesitan provienen de un PIM. Las trasformaciones de CIM a PIM y de PIM a PSMs deben definirse, lo cual es una tarea difícil y especializada. Pero esto sólo debe ser definido una vez y se puede aplicar en el desarrollo de muchos sistemas. Por el simple hecho de definir una buena transformación se aumenta la productividad, se mejora el tiempo de desarrollo, se optimiza el tiempo de mantenimiento, etc, pero sólo puede ser realizada por ingenieros del software.

La portabilidad se consigue con el desarrollo de los PIM, los cuales son independientes de plataforma. El mismo PIM puede ser transformado en múltiples PSMs para diferentes plataformas. Todo lo que se especifique en el PIM es completamente portable.

La extensión para cada plataforma se puede conseguir utilizando las herramientas de transformación automática. Para plataformas populares hay un buen número de herramientas, pero para las menos conocidas hay muy pocas.

Además, la evolución y la documentación son mucho más mantenibles debido a que muchas herramientas son capaces de mantener las relaciones de trazabilidad entre el PSM y el PIM. Por ello, cualquier cambio que se realice en el PIM se refleja en el PSM y la documentación sigue siendo consistente.

2.2.3.4 Lenguajes de modelado en MDA

El enfoque MDA está definido por la OMG. Ésta define un conjunto de lenguajes de modelado para, entre otros propósitos, dar soporte a MDA. Uno de los lenguajes más conocidos es UML, ya que su utilización esta ampliamente extendida en el ámbito de la ingeniería del software.

Los lenguajes utilizados en MDA necesitan tener definiciones formales para que las herramientas sean capaces de transformar los modelos automáticamente escritos en esos lenguajes. Entre estos lenguajes, también se encuentra MOF (*Meta Object Facility*) , el cual se utiliza para definir el resto de lenguajes, como UML. De esta forma, MOF asegura que las herramientas sean capaces de leer y escribir en todos los lenguajes estandarizados por la OMG, ya que todos están basados en MOF.

Las definiciones de transformaciones dentro de MDA se describen utilizando el lenguaje estándar que la OMG ha definido para escribir las transformaciones entre modelos. Este lenguaje estándar se denomina QVT (Query/View/Transformations).

- **UML como lenguaje de PIM**

UML es un lenguaje de modelado genérico que permite modelar el comportamiento de un sistema software independientemente de la plataforma en la que vaya a ser implantado. Esto permite que a partir de modelos UML se puedan generar PSMs. Sin embargo, UML tiene algunos puntos débiles en el comportamiento y en la parte dinámica que no nos permiten generar un completo PSM a partir de un PIM. UML incluye diagramas para modelar el comportamiento, pero no es un lenguaje formal y esto hace que se creen ambigüedades en el proceso de transformación.

➢ Meta Object Facility (MOF)

La especificación de la OMG de MOF nos permite distinguir claramente entre tipos e instancias de una forma apropiada y elegante. Tiene como objetivo permitir la interoperación de las herramientas de metamodelado, y de las que dan soporte al proceso de desarrollo software basado en MDA. MOF es una arquitectura de metamodelado que consta de cuatro capas.

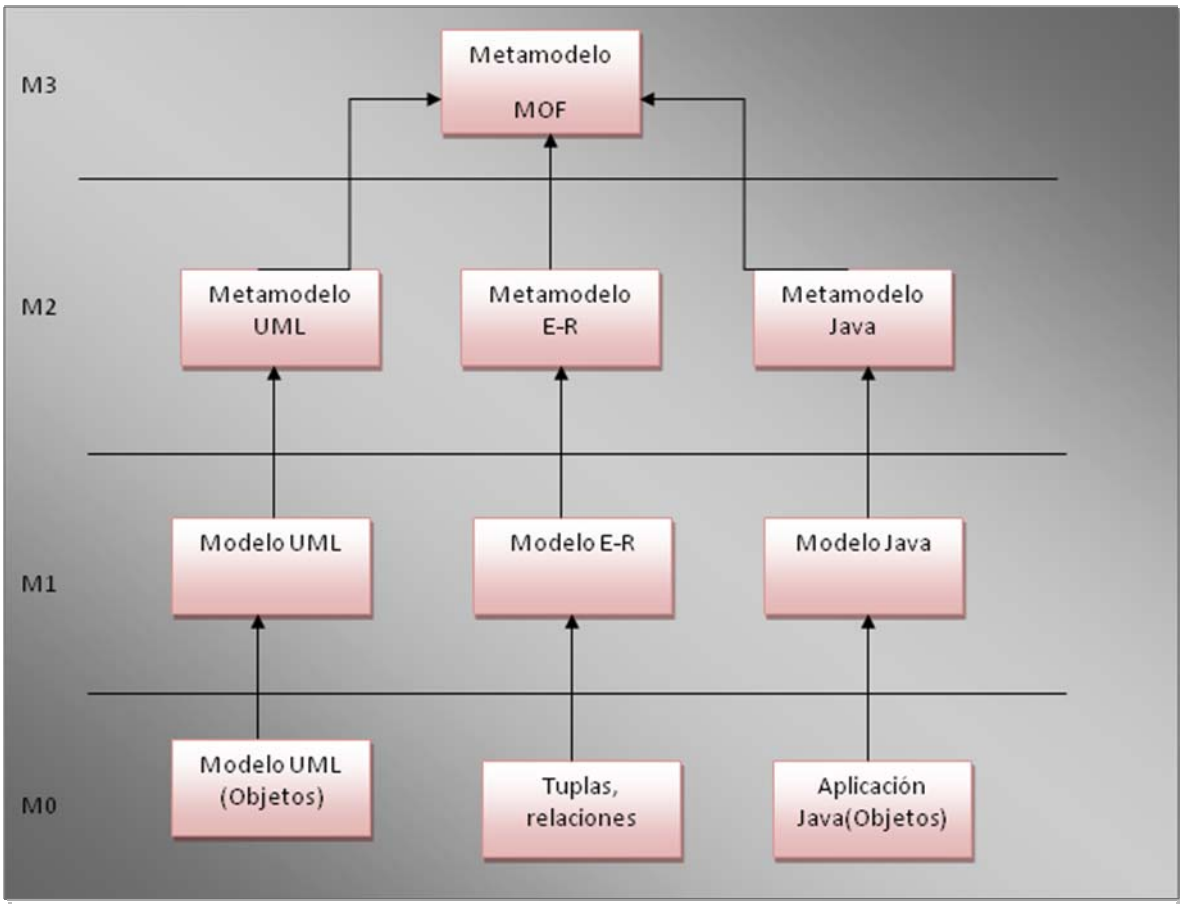


Figura 5: Torre Reflexiva de MOF

Un ejemplo de la torre reflexiva de MOF se muestra en la figura 5. . La capa superior (M3) es la más abstracta. Esta capa define el lenguaje abstracto utilizado para describir entidades de la capa colocada debajo (metamodelos). Las especificaciones de MOF proponen el modelo MOF como el lenguaje abstracto para definir todo tipo de metamodelos, como UML. En el nivel M2 están definidos los tres metamodelos de UML, E-R y Java. Todos ellos están definidos

conforme a MOF. En este ejemplo también se pueden distinguir los PIM y los PSM. Los modelos UML y E-R son PIM, en cambio, el programa Java es un PSM.

La arquitectura de metamodelado MOF está acompañada de otros dos estándares, XMI y QVT. El primero es un esquema XML para la representación de modelos MOF, y el segundo es el lenguaje estándar de transformación de modelos MOF.

En la figura 6 se resume el capítulo mediante una torre MDA. Como ya se ha explicado, en la primera capa se enclava las especificaciones MOF, la siguiente capa consiste en los metamodelos UML y Java. Se interrelacionan mediante una flecha bidireccional para considerar que pueden hacer mapping bidireccional. Después del metamodelo econtramos los modelos de cada nivel superior y por último la aplicación correspondiente a cada caso.

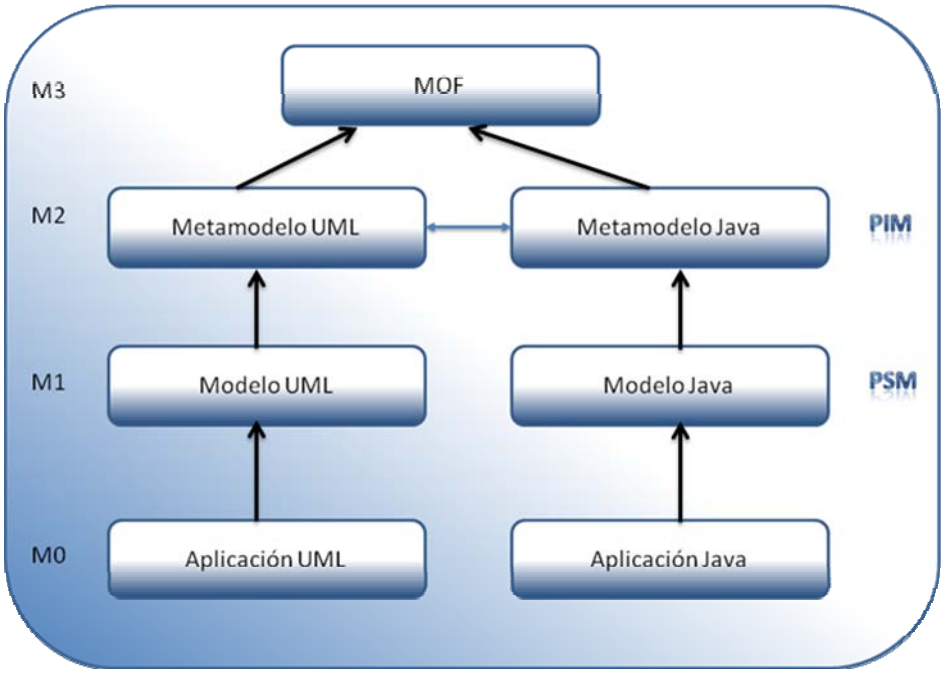


Figura 6: Torre MDA

2.3 Conclusiones

Debido a la gran flexibilidad y a la solución que propone MDD a la problemática de la complejidad de sistemas software actual, es posible afirmar que MDD supone un gran avance en el mundo de la informática. El nivel de abstracción alcanzado con las herramientas derivadas de MDD mejora la cohesión entre las “piezas” que trata de encajar el ingeniero del software, haciendo el desarrollo software mucho más versátil y amigable. El tiempo invertido en el mismo decrece y las curvas de vida del software varían a mejor. Todo ello nos lleva a considerar esta nueva dirección como “la dirección”, la dirección del futuro.

3. Herramientas de gestión de modelos

El uso de modelos para desarrollar software nos da soluciones que son independientes de la tecnología, cuyo código fuente se obtiene a través de múltiples transformaciones y de generaciones de código para diferentes tecnologías y lenguajes de programación. Esta forma de trabajo es conocida como Gestión de Modelos *(Model Management*). La gestión de modelos es una forma de trabajo, generalmente aplicada a MDD, que se presenta como un avance en el área y que está muy aceptado por la comunidad de ingenieros. De ahí que en muy poco tiempo haya una amplia gama de herramientas que le dan soporte.

- AMMA (Atlas Model Management Architecture) es una plataforma de gestión de modelos diseñada y desarrollada por INRIA. Esta plataforma es un plug-in de Eclipse. Está basada en el estándar de la OMG, MOF, y utiliza Ecore como lenguaje de modelado. MOF propone el modelo MOF como lenguaje abstracto para definir todo tipo de modelos y "arquitectura de nivel cuatro" que está enfocada a MDA. AMMA proporciona ATL (*Atlas Transformation Language*), un lenguaje declarativo e imperativo (*lenguaje híbrido*) para la transformación de modelos. A pesar de que no está basado en el estándar QVT, se utiliza actualmente en numerosos grupos y proyectos de investigación, e incluso hay proyectos que han utilizado ATL, como el proyecto *ModelWare*.

- XMI-XSLT es un marco de trabajo tecnológico que no está basado en QVT y proporciona soporte para el manejo de modelos. El uso de esta herramienta permite la transformación de modelos definiendo los modelos como documentos XMI y

utilizando XSLT para transformar de un documento a otro. El problema que tiene este marco de trabajo es que la transformación está basada en sintaxis y no en semántica.

- Borland Together es un producto que integra Java IDE, que originalmente nace de JBuilder con una herramienta de modelado UML. Tiene soporte para Eclipse y los diagramas pueden crearse de forma importada. Genera soporte para UML, Java 6, C++ y CORBA.
- SmartQVT es una implementación completa en java del lenguaje operacional QVT. La herramienta se distribuye como un complemento a Eclipse de libre distribución y se caracteriza por estar implementada de forma imperativa
- openArchitectureWare es un generador de entornos de trabajo modular implementado en Java. Esta herramienta ofrece soporte a la trasformación de un modelo a otro, de texto a modelo, y de modelo a texto. Está basado en la plataforma Eclipse y soporta modelos basados en EMF pero puede trabajar con otros modelos.
- MediniQVT es una herramienta de transformación de modelos. MediniQVT está implementado para realizar las transformaciones que se instauraron desde la OMG, es decir, siempre basando sus transformaciones de modelos en QVT. La aplicación incluye herramientas para el desarrollo de transformaciones, así como un depurador gráfico y un editor. Estos componentes son de libre distribución pero sólo para uso no comercial. MediniQVT efectúa las transformaciones QVT expresadas en la sintaxis concreta del lenguaje de relaciones de QVT (*QVT relations*). MediniQVT está integrado en Eclipse. La aplicación posee un editor con asistente de código que permite especificar las relaciones modelo origen modelo destino, ya sea el elemento o sus propiedades. MediniQVT proporciona también un depurador para efectuar las transformaciones y de esta forma evitar errores en el modelo destino. Un hecho a destacar es que permite las transformaciones bidireccionales.

- GME (Generic Modelling Environment) es un conjunto de herramientas configurables para crear modelos de diseño específico. Para crear uno de dichos modelos es importante tener en cuenta que el peso de la correcta definición del dominio recae sobre el metamodelo, es decir, en la configuración del metamodelo se debe modelar el dominio correcto de la aplicación. El metamodelo de entrada debe contener toda la sintaxis, semántica e información del dominio; conceptos que serán utilizados para construir modelos, ver qué relaciones pueden existir entre ellos, cómo pueden estar organizados y cómo los verá el ingeniero de modelos. El metamodelo también debe contener las reglas que gobiernan la construcción de modelos.

  El lenguaje de metamodelado está basado en los diagramas de clases UML y en las restricciones OCL. Lo metamodelos que especifican el modelo se utilizan para generar automáticamente un entorno de dominio específico. Este entorno se utiliza después para construir modelos específicos de dominio que se almacenan en una base de datos de modelos o en formato XML. Estos modelos se utilizan para generar automáticamente las aplicaciones.

  GME tiene una arquitectura modular y extensible. GME es fácilmente extensible; los componentes definidos fuera del ámbito de la herramienta y utilizados para la extensión de la misma pueden escribirse en cualquier lenguaje que soporte COM (C++, Visual Basic, C#, Python etc.). GME tiene muchas características avanzadas y dispone de un elemento ya integrado que impone todas las restricciones de dominio durante la construcción de los modelos, denominado "*director*". GME soporta múltiples aspectos del modelado como son la combinación de lenguajes de modelado y el soporte de bibliotecas de modelos para su reutilización.

- GMF (Graphical Modelling Framework) proporciona un generador de modelos. GMF está embebido en Eclipse, plataforma de desarrollo que permite la ejecución de los

editores gráficos generados por GMF a partir de un modelo. Eclipse está construido y trabaja en Java. GMF está basado en EMF (*Eclipse Modelling Framework*) y en GEF (*Graphical Eclipse Framework*), ambos proporcionados por la plataforma Eclipse. En GMF se ha adoptado el término "*toolsmith*" para referirse a los desarrolladores que utilizan la herramienta para construir extensiones que posteriormente se integran como parte de la misma. Estas extensiones reciben el nombre de plug-ins. Otro término, "*practitioner*", se utiliza para referirse a aquellos que utilizan los plug-ins como medio para el desarrollo. Durante la utilización de GMF para la generación de un modelo, la descripción del modelo se realiza una sola vez, al comienzo. Una vez realizada la especificación del dominio, la herramienta se encarga de interpretar las correspondencias con el modelo durante el resto de proceso de generación del editor.

3.1 GMF

Graphical Modelling Framework (GMF) proporciona una herramienta para la generación de editores gráficos basados en EMF y GEF. Una característica que hace destacar a GMF es la reutilización de la definición gráfica para diferentes dominios y aplicaciones, esto es que para diferentes conceptos se pueden reutilizar las metáforas gráficas ya definidas en GMF para dichos dominios y aplicaciones. Esta característica se consigue modelando por separado las componentes gráficas que se corresponden con cada uno de los elementos del dominio y la definición de la paleta de herramientas, la cual tendrá una herramienta por cada primitiva. Para terminar, GMF proporciona una definición de mapping o correspondencia mediante la que se asocia cada primitiva de modelado con su componente gráfica y con su herramienta dentro del editor que se está generando.

GMF está soportado por Eclipse, lo cual es una ventaja dado el interfaz amigable de esta plataforma y su uso extendido. A pesar de que GMF parece bastante intuitivo debido a una interfaz gráfica rica en colores y formas, hablando en términos de desarrollo no lo es tanto. Esto se debe a la gran complejidad que envuelve a la herramienta y la poca ayuda que se puede extraer de la misma. Por el contrario, tiene bastantes ventajas, como por ejemplo, la automatización de pasos a seguir en el ciclo de desarrollo software y la garantía de un producto final libre de errores de compilación, ya que el proceso es guiado y supervisado por la propia herramienta.

GMF se compone de varias partes que deben coexistir y encajar para conseguir el software deseado. En la secuencia de creación se distinguen la definición del metamodelo, la generación de código, la definición gráfica, la definición de las herramientas del modelo, la correspondencia entre los elementos antes citados y, por último la generación de la herramienta (ver Figura 7).

Como se puede observar en el diagrama de la Figura 10, el centro del proyecto es el *Domain Model*, es decir el modelo del dominio o metamodelo que será el origen del proceso y del cual se derivarán el resto de pasos, el cual se define utilizando EMF. EMF utiliza un lenguaje de definición de modelos denominado Ecore. Todos y cada uno de los pasos de GMF están relacionados con el modelo y como indica su nombre, Ecore, constituirá el centro del proyecto en todo momento.

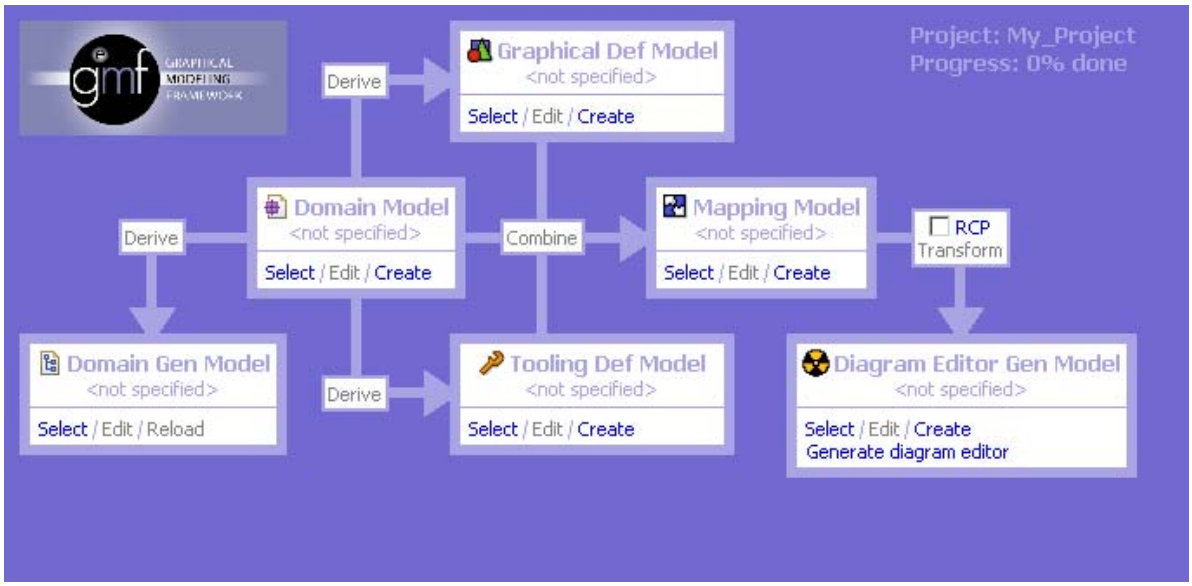


Figura. 7 Diagrama del proyecto

3.1.1 EMF

GMF precisa de un metamodelo y para ello se sirve de EMF (Environment Modelling Framework), que se trata de un marco de modelado que soporta y genera documentos XMI con un esquema XML Ecore con la especificación del dominio. Cabe apuntar la versatilidad de EMF, que permite definir modelos – se entiende que son diagramas de clases de pseudo-UML – para la implementación del modelo concreto que se desea crear a partir de la herramienta genérica EMF, tanto de forma textual en sintaxis XMI (ver Figura 8), como de forma gráfica, mucho más intuitivo y visual (ver Figura 9). Esta última característica hace el conjunto de herramientas EMF-GMF especial y novedoso, a la par que cómodo, relativamente fácil y cercano al desarrollador. EMF mantiene el grado de abstracción con respecto a la plataforma, sin embargo no complica la creación de modelos dependientes de plataforma como ocurría con las herramientas CASE, destacadas por su complejidad que iba ligada a la de la plataforma en la que estaban soportadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore">
  <eClassifiers xsi:type="ecore:EClass" name="Clase">
    <eStructuralFeatures xsi:type="ecore:EReference" name="Contiene" lowerBound="1"
        upperBound="-2" eType="#//Atributo" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Atributo"/>
</ecore:EPackage>
```

Fig. 8 Captura del modelado EMF en su versión texto

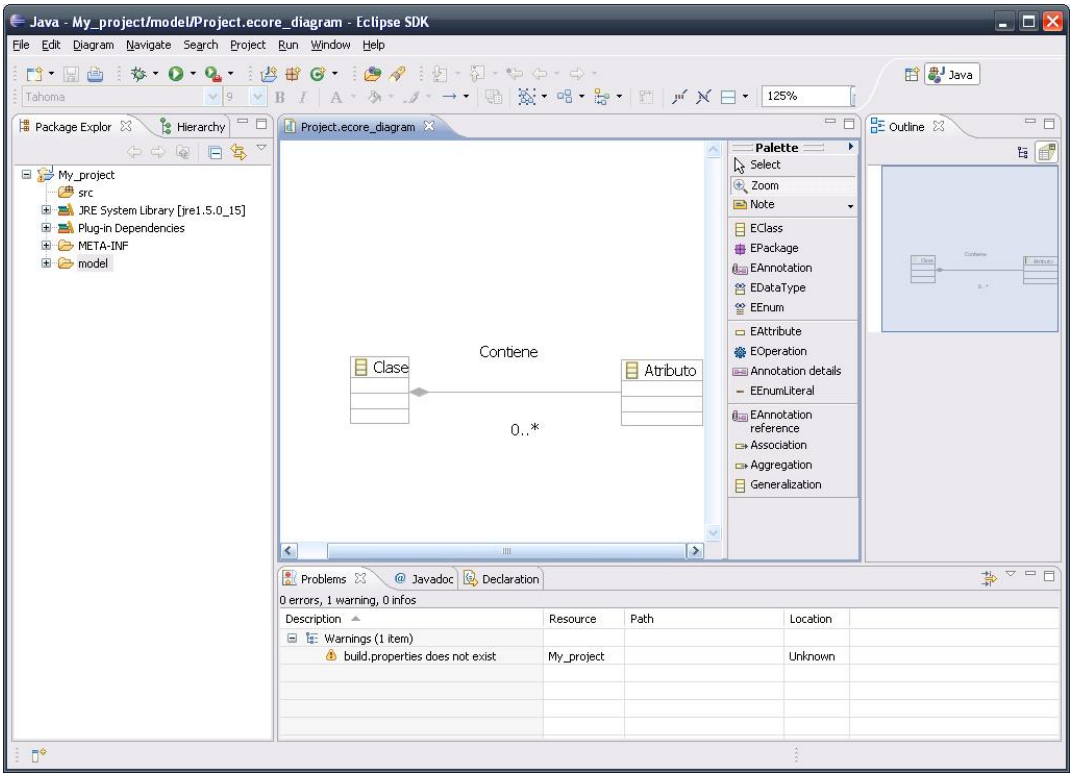


Fig. 9 Captura del modelado EMF en su versión gráfica

Como indica la figura 6, se necesita un modelo de entrada a GMF que se especifica en EMF. Por lo tanto, lo primero al iniciar el proceso de modelado es trasladar el modelo o metamodelo a EMF en lenguaje Ecore. EMF, por tanto, proporciona el elemento que GMF necesita como entrada, un modelo en lenguaje Ecore, al que a partir de ahora se denominará como modelo Ecore. El modelo Ecore se crea a través de un editor gráfico que viene integrado en EMF. Se trata de una herramienta que consta de una paleta y una hoja de modelado que mediante "drag and drop" permite construir el modelo de entrada a GMF. Al finalizar el modelo, éste se guarda y se actualiza el documento XMI asociado, el cual está construido mediante un esquema XML de tipo Ecore. Esto se debe a que el modelo tiene dos representaciones, una de tipo texto y otra de tipo gráfico y al estar relacionadas se mantienen actualizadas constantemente. En GMF se va a desarrollar un editor de modelos a partir de un metamodelo, y por tanto el modelo Ecore debe representar el modelo que se haya escogido para la implementación del editor y de esta forma en el editor se recogerán los elementos que se hayan especificado en el modelo. Dicho modelo se especifica mediante relaciones y clases que contienen el nombre, los atributos, las operaciones que pueda realizar y las anotaciones para esa clase en caso de que se quieran hacer (estas últimas no tendrán especial relevancia en el resultado final). Todos y cada uno de los elementos pueden tener tantos atributos y operaciones como el desarrollador quiera asignar, es decir, si se quiere crear una clase sin atributos ni operaciones, se puede realizar. Dentro de la paleta disponible para la edición del documento Ecore (ver Figura 10) es posible encontrar diversos elementos para incluir en el diagrama Ecore que se este modelando. Entre ellos se encuentran las clases (elemento *EClass*) que representan los distintos elementos que tendrá el editor a construir, los paquetes (elemento *EPackage*) que son elementos que contienen más clases y paquetes, los atributos(elemento *Eatribute*) que se corresponden a las características que distinguen a unas clases de otras, las operaciones (elemento *Eoperation*) que indican las operaciones que hace

cada clase y por último tenemos las relaciones de asociación (*Association*) y agregación(elemento *Aggregation*) que se definen especificando su nombre y cardinalidad.

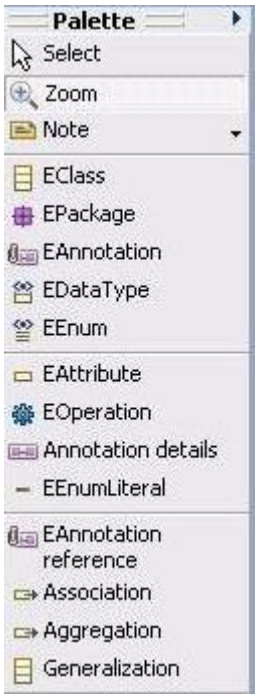


Fig. 10. Paleta de edición de EMF

EMF tiene reglas estrictas para la creación del modelo que vienen impuestas por las restricciones que tiene GMF para la lectura de documentos Ecore. En la raíz del diagrama debe colocarse la clase que vaya a contener todo el modelo y que después tendrá la función de "*canvas*" en la definición gráfica de GMF. Todos aquellos elementos que se quieran hacer visibles en la herramienta a crear se deben unir a la raíz con una relación de agregación, es decir, la aplicación contiene a todos sus elementos (ver Figura 11). Las relaciones que se hayan definido en el modelo entre los elementos se pueden realizar mediante relaciones de asociación ya sean inclusivas o no inclusivas. Todas ellas se definen con un nombre y una cardinalidad máxima y mínima. Cabe apuntar que la cardinalidad máxima N se traduce en un -1 en el diagrama EMF. Por ejemplo, en la figura 11 se puede observar como ejemplo que el

canvas contiene a los dos elementos, y por tanto es la raíz del modelo. Element1 y Element2 serán elementos del modelo que se pueden asociar entre sí con diferentes cardinalidades. Por lo tanto, El elemento Raíz contiene dos Elementos que se pueden asociar entre sí. El modelo Ecore vinculado al diagrama se actualiza automáticamente en sus dos representaciones (gráfica y textual).

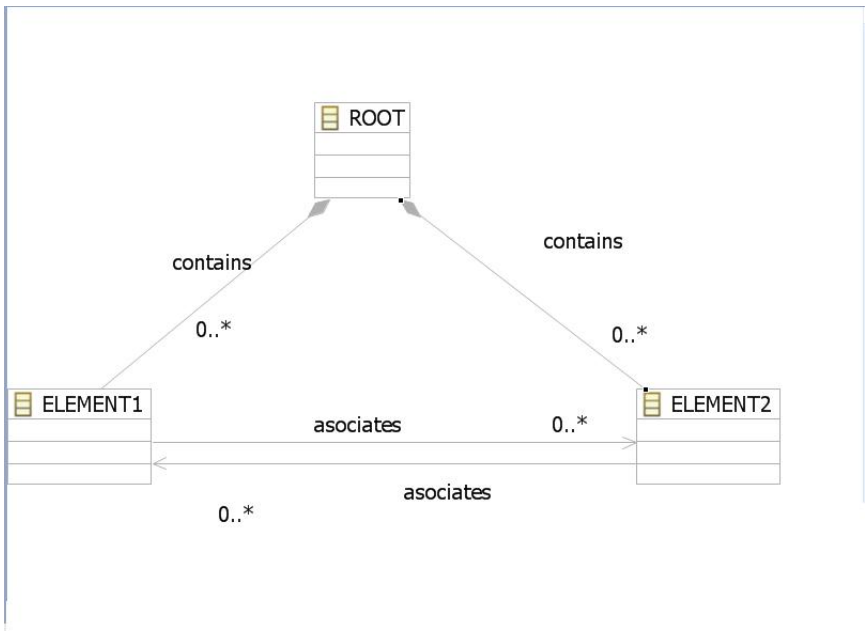


Figura 11. Árbol de la definición de un metamodelo en EMF

Con EMF es posible crear un elemento con extensión .genmodel. Este elemento permite generar automáticamente el código y las clases Java que después se utilizarán para la creación del editor que se esta definiendo a partir del documento Ecore. La generación automática es una labor que resulta valorable, ya que siguiendo unos patrones, GMF permite al ingeniero una abstracción aún mayor del lenguaje de programación generando automáticamente todo el código perteneciente a las clases del editor. De esta forma, el desarrollador se abstrae completamente del código y se

puede dedicar plenamente a la labor de modelado y la generación de componentes gráficas. Además, el componente genmodel contiene una característica adicional que permite la comprobación de la validez del metamodelo creado, lo cual asegura que el modelo tiene cierto grado de consistencia y que no se van a arrastrar errores de compilación en el proceso de creación del editor.

3.1.2 Definición Gráfica

Volviendo a la figura 7 y una vez definido el dominio del modelo, se puede empezar a diseñar la definición gráfica de las primitivas de modelado. La definición gráfica consiste en decidir qué primitivas de modelado harán la función de nodos (elementos del editor), cuáles serán conectores (enlaces entre los elementos del editor) y cuáles etiquetas (propiedades de los elementos y enlaces del editor). Esta información debe ser coherente y definitiva, de otra forma, se deberá comenzar el proceso de generado desde el comienzo, aunque sean cambios pequeños de última hora. La definición gráfica es la representación de los elementos del modelo en el nuevo entorno que se está creando.

En la especificación gráfica GMF ofrece numerosas posibilidades para determinar la forma y color de las primitivas de modelado lo cual supone una enorme ventaja hablando en términos de diseño, accesibilidad y amigabilidad con el presunto entorno a crear.

3.1.3 Definición de herramientas

Continuando con el gráfico de pasos de GMF propuesto en la figura 7, después de la especificación gráfica, se debe realizar la creación y especificación del panel de herramientas. Consideramos panel de herramientas como el panel lateral del editor a crear que proporcionará un funcionalidad “drag and drop”, mostrándonos las primitivas de modelado en

la forma que se han diseñado en la definición gráfica. En este paso también se definen los iconos del editor. Siguiendo la definición del modelo elegido y mediante un método de asociación de nodos, enlaces y etiquetas, la elección de las herramientas que conforman el modelo se debe realizar manteniendo la coherencia con el paso anterior y preservando la integridad del proyecto. Siguiendo con el ejemplo ilustrado en la figura 11, en este paso se debe decidir qué botón del panel de herramientas del editor que se está creando se corresponde con Element1 y cuál con Element2. Este paso se corresponde con las utilidades que tendrá el nuevo entorno y las herramientas que estarán visibles y disponibles en la paleta.

3.1.4 Correspondencia de elementos

Dentro del contexto de la Figura 7, después de hacer la especificación de las herramientas, a continuación se debe hacer la correspondencia o mapping de elementos. La correspondencia entre las primitivas e modelado, la definición gráfica de esas primitivas y la especificación de herramientas es imprescindible para la integridad del modelo. En este estadio todo lo creado anteriormente cobra un sentido, es decir, se produce la correspondencia de las metáforas gráficas con las herramientas de creación y operación. De esta forma al finalizar este paso, se dispone de un *.gmfmap* que relaciona y aúna todos y cada uno de los elementos creados con anterioridad. A partir de este proceso se crea un generador en el que se pueden definir diversas cualidades del editor, así como seleccionar la extensión del archivo de salida y además hace un generado automático del editor especificado a través de todos los anteriores pasos y permite una recompilación comprobando la integridad de la herramienta creada.

3.1.5 Generación del diagrama

Este paso es el último del proceso de creación. Puede ser que aunque sintácticamente todo esté correctamente especificado y aunque los pasos se hayan realizado adecuadamente el

resultado puede no ser el que se esperaba, es decir, que lo que se ha modelado no se corresponde con lo que se esperaba. No sólo importa la sintaxis y seguir los pasos de forma adecuada, sino utilizar una semántica adecuada. La creación del generador de la herramienta sin errores indica que todas las anteriores especificaciones se han realizado correctamente y mantienen coherencia. Como ya se ha dicho anteriormente puede que la herramienta no se comporte como se esperaba pero hay una gran probabilidad de que esté libre de errores.

4. Descripción del metamodelo de operaciones del sistema

Uno de los problemas asociados con la ingeniería de sistemas complejos es identificar el conocimiento relevante para la construcción del sistema. Las operaciones del sistema son una parte esencial de este conocimiento, sin embargo no se proporcionan guías específicas para el modelado de las mismas. Este capítulo está basado en el diseño definido en Alarcón et al.

4.1 Operaciones de un sistema

Las operaciones de un sistema se entienden con el conjunto de interacciones que actúan en el sistema, entradas al sistema y diferentes tipos de salidas, clasificadas en respuestas, notificaciones y alarmas. Se entiende sistema como el conjunto de software y hardware, con una interfaz que permite la interacción con el mismo. En la figura 12 se muestra el nivel de interacción 1, que representa el mayor nivel de abstracción relativo al aspecto de las operaciones del sistema, las cuales se reducen a entradas y salidas del sistema entre el sistema y el operador. Entendemos que el operador no interactúa directamente con el sistema, sino que lo hace mediante una aplicación externa que tiene la función de interfaz estableciendo la interacción física con el sistema.

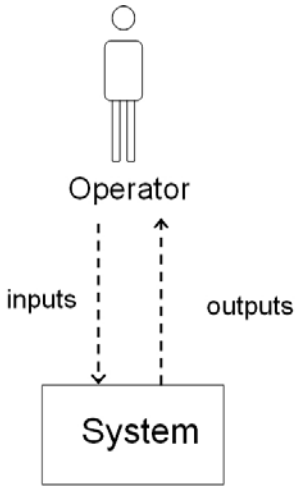


Figura 12 : Nivel de interacción 1

La figura 13 muestra el segundo nivel de abstracción en la interacción entre sistema y operador. En este nivel, se incluye una aplicación software externa al sistema a la que llamamos OperatorFrontend, que posibilita el conjunto de interacciones entre operador y sistema.

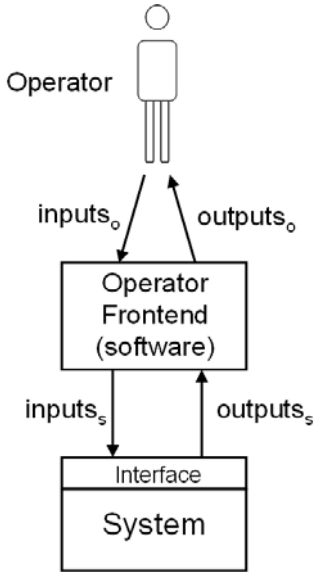


Figura 13: Nivel de interacción 2

Estas interacciones se pueden modelar pero sólo tendrán sentido si, y sólo si, existe al menos una variable del entorno del sistema que tenga que ser controlada, monitorizada o ambas cosas por el sistema. Intuitivamente, podemos considerar que un modelo de operaciones de un sistema representa el conjunto de interacciones de ese sistema con una aplicación externa. Este conjunto incluye interacciones de entrada al sistema enviadas por un operador externo e interacciones de salida enviadas desde el sistema al exterior. Las variables monitorizadas están relacionadas con fenómenos que están en el entorno del sistema y cuyos valores están siendo observados por el sistema, como por ejemplo la temperatura exterior. Las variables controladas son aquellas inherentes al sistema, y cuyo valor puede ser controlado o cambiado por el propio sistema, como por ejemplo, la temperatura en el interior de un coche. Por último

hay variables que necesitan ser monitorizadas y controladas por el sistema, como la temperatura del agua de un radiador.

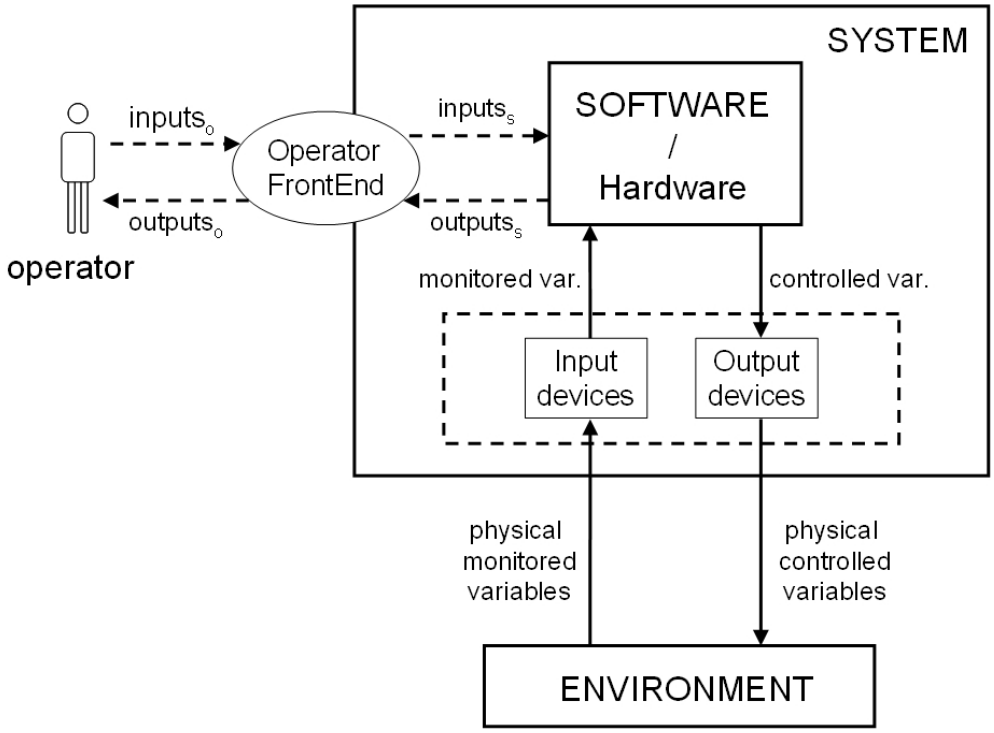


Figura 14: Nivel de interacción 3

En la figura 14, se muestra un modelo del sistema que representa el nivel de interacción 3. Como se puede apreciar, el usuario es capaz de interactuar con un *FrontEnd* o software de interacción con el sistema mediante operaciones de entrada, y a su vez, el sistema es capaz de interactuar con el Frontend enviando salidas del sistema en forma de respuestas a las operaciones recibidas y avisos de cambios en las variables que el propio sistema monitoriza y controla. Es necesario destacar que el sistema consta de variables a monitorizar, controlar o ambas, y que como se puede apreciar en la figura 14, existen dos tipos de variables, las propias del entorno y las variables internas del sistema. A través de este planteamiento, se ha construido un metamodelo de operaciones que recoge estas ideas. Las variables internas

representan valores discretos de las variables del entorno. Por ejemplo, un sensor capta un valor continuo de la temperatura y lo transforma.

4.2 Operaciones de un sistema

Aunque los documentos de especificación de requisitos de operación y de sistemas de operación han tenido relevancia en la comunidad de la ingeniería del software, no ha sido el caso del modelado de operaciones. Como primera aproximación, se definió el modelo de operaciones de un sistema como el conjunto de entradas y salidas que permite la interacción con el sistema. Por lo tanto, el modelo de operaciones se consigue mediante la unión de dos conjuntos de operaciones, el de entrada (I) y el de salida(O).

$$MOS = \{I \cup O\}$$

Donde:

- Entradas (I): es un conjunto que representa las operaciones de entrada del sistema y pueden ser de dos formas:
    - Los comandos representan el conjunto de entradas aceptadas por el sistema. Cada elemento del conjunto corresponde a una de las operaciones admitidas por el sistema, lo que en términos de modelo de operaciones llamaremos operaciones.
    - Otro conjunto de entradas del sistema existente corresponde a las entradas no aceptadas y no reconocidas por el sistema. Sin embargo, este tipo de entradas no aporta nada al modelo de operaciones del sistema y por lo tanto no las tendremos en cuenta.

Por lo que las entradas del sistema corresponderán exclusivamente a los comandos de operación, {I =C}.

- Salidas (O): es un conjunto que representa las repuestas a operaciones recibidas y avisos producidos por el sistema. Pueden ser de tres formas:
    - Respuestas: La salida se produce como respuesta a la ejecución de un comando de operación. Todo comando de operación aceptado por el sistema generará una respuesta. Consideraremos que un comando de operación aceptado por el sistema estará asociado a un solo elemento del sistema y por lo tanto una respuesta corresponderá a un solo elemento y comando.
    - Notificaciones: una notificación es información enviada por el sistema, relativa al cambio de estado producido en una variable monitorizada o relativa a un evento producido internamente durante la ejecución del sistema. Una notificación no corresponde a la respuesta a un comando de operación, pero se produce durante el transcurso del funcionamiento del sistema, resultando del cambio de estado de una de las variables que monitoriza el sistema.
    - Alarmas: Un tipo especial de notificación está constituido por el conjunto de alarmas del sistema, y que puede estar producido durante el funcionamiento normal del sistema o debido a fallos detectados en el mismo. Una alarma es una notificación con prioridad establecida, y representa una situación crítica o que requiere la máxima atención por parte del operador.

4.3 Metamodelo de operaciones de un sistema

Todo sistema operable admite un conjunto de operaciones enviadas desde el exterior, y genera hacia éste salidas en función de una única operación recibida o de una secuencia de operaciones recibidas a lo largo del tiempo. De esta manera, el operador puede interaccionar con el sistema mediante la inicialización de operaciones de entrada que pueden provocar cambios y modificaciones en el propio sistema, y así se obtendrán reacciones del sistema, es decir, operaciones de salida. Alarcón define el modelo de operaciones en Alarcon et al. como el conjunto de entradas al sistema, respuestas del sistema a esas entradas y avisos generados por el sistema (ver fig. 15).

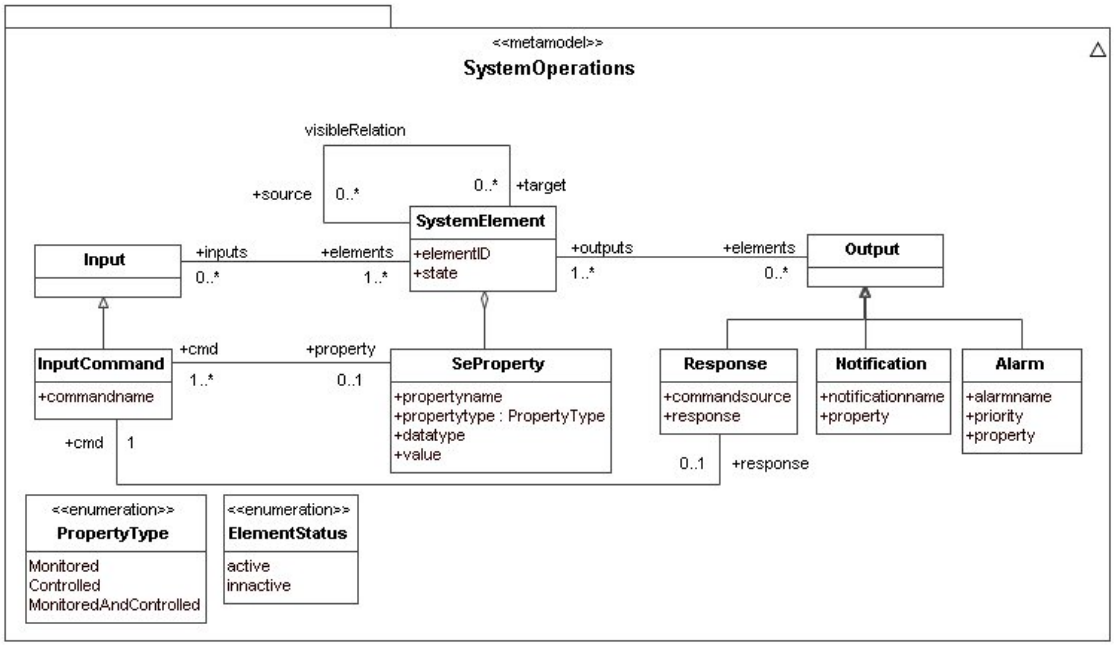


Figura 15. Metamodelo de operaciones del sistema

4.4 Elementos de un sistema de operaciones

El metamodelo de operaciones del sistema detallado en la figura 11 contempla las operaciones o comandos de entrada al sistema como las notificaciones y alarmas que el sistema produce.

1. Parte del sistema perceptible:
    - **a.** *SystemElement*: representa los diferentes tipos de elementos que se pueden tener, los cuales tienen capacidad de interacción con el exterior y que componen un sistema. Cada elemento del sistema estará definido por un nombre y un identificador único, su estado desde el punto de vista de las operaciones y una serie de reglas de conexión que determinarán las conexiones válidas entre elementos
        - **i.** *State*: Representa el estado del elemento desde el punto de vista de las operaciones.
        - **ii.** *Id*: Será el identificador único del elemento.
    - **b.** *SEProperty***:** Representa las propiedades que relacionan a un elemento con su entorno. Cada propiedad viene identificada por un nombre o identificador único y el tipo de la propiedad, es decir, monitorizada, controlada o monitorizada y controlada.
        - **i.** *PropertyName:* Es el nombre de la propiedad.
        - **ii.** *PropertyType:* Monitorizada, controlada o ambas.
        - **iii.** *DataType:* El tipo de la propiedad
        - **iv.** *Value:* Valor de la propiedad.
    - **c.** *VisibleRelation:* Asociación o conexión visible desde fuera del sistema que puede ser establecida entre los distintos tipo de elementos del sistema.
2. Parte del sistema no perceptible:
    - **a.** Operaciones de entrada: acciones de entrada enviadas desde el exterior a elementos del sistema. Es importante saber que una operación puede modificar una propiedad.
        - **i.** *CommandName:* Nombre de la operación de entrada.

b. Operaciones de salida: representa las acciones de salida producidas por los elementos del sistema a una aplicación externa, incluye notificaciones y respuestas

i. *Response:* representa la respuesta del sistema frente a un comando de operación de entrada aceptado. Una aplicación siempre tiene su origen en un único comando, ahora bien, un comando puede tener diferentes respuestas.

1. *CommandSource:* Es el comando al que se asocia este resultado.

2. *Response:* Contenido de la respuesta.

ii. *Notification:* representa las notificaciones del sistema debido a cambios en variables monitorizadas y/o controladas por el propio sistema. No están relacionadas con comandos de operación aceptados por el sistema.

1. *NotificationName:* Nombre del cambio.

2. *Property:* Propiedad modificada.

iii. *Alarm:* representa las notificaciones del sistema debido a cambios en el estado de variables monitorizadas por el propio sistema y que requieren especial atención por parte del operador.

1. *AlarmName:* Nombre de la alarma.

2. *Pririty:* Prioridad.

3. *Property:* Propiedad afectada en la alarma.

5. Espora

En este capítulo se expondrá la implementación de una herramienta basándonos de un modelo de operaciones. Esta herramienta, denominada Espora, permite la creación de modelos en xml con un interfaz y una implementación completamente gráfica. La aplicación está completamente creada en GMF, que en el capítulo 3 ya se explicó que consiste en una herramienta de definición automática de modelos a partir de un modelo de operaciones.

5.1 Implementación

En este capítulo se va a tratar el proceso de implementación de la herramienta Espora , que permite la definición de modelos, generada a través de GMF. La herramienta Espora ha sido realizada íntegramente desde el entorno de GMF embebido en Eclipse. El objetivo de esta aplicación es proporcionar un entorno que permita la definición de modelos a través de un interfaz completamente gráfico, intuitivo y accesible.

GMF, como ya se ha citado anteriormente, permite la definición de un modelo de forma completamente automática. Con sólo especificar las opciones correctas en el formato adecuado, GMF crea una aplicación exclusivamente para formar una instanciación o elemento conforme a las primitivas definidas en el modelo inicial. Espora es un ejemplo de ello.

Espora es una herramienta de definición de modelos de operación que lleva asociada una componente gráfica predefinida a cada elemento del modelo. Esta desarrollada de forma absoluta en GMF y su creación ha sido completamente automática, es decir, no se ha modificado ni incluido nuevo código que el que aporta GMF.

La definición de Espora se ha realizado en los pasos que ya se comentaron en el capítulo 4. La siguiente explicación de pasos no sólo resume de forma escueta lo que se realiza en cada paso, sino también explica en detalle la forma en la que se realizó y el porqué de esta forma.

5.1.1 Definición del modelo en EMF.

Para la implementación de cualquier aplicación en GMF se necesita un modelo de operaciones de entrada que nos proporcione unas primitivas de modelado para la aplicación en concreto. El modelo correspondiente a la herramienta Espora consiste en Et.Alarcón, explicado en el capítulo 4. Por razones de usabilidad y accesibilidad, y para hacer la aplicación mucho más intuitiva al usuario se decidió hacer diversas modificaciones con respecto al modelo inicial. A la clase *InputCommand* le ha sido cambiado el nombre por *Operation*. *Response*(Si una *Operation* genera respuesta) se ha decidido suprimir y se le ha añadido a *Operation* un atributo de tipo booleano (atributo *response*). Dentro de la clase *SystemElement* existe un identificador único y un estado en el modelo Et.Alarcón, pero se ha decidido que en su lugar se prefiere poner un nombre sólo, ya que GMF genera identificadores únicos automáticamente, y suprimir el atributo *state*. La relación de *Alarm* y *Notification* con *SystemElement* se ha sustituido por *Triggers* y *Sends* respectivamente, y la de *Operation* se define por *Executes*. Asimismo, la relación de *Porperty* con *Alarm* y *Notification* se transforma en *Notifies* y *Generates* respectivamente y aparece la relación *Has* entre *SystemElement* y *Property*. (Ver Fig. 16)

Durante la creación de este diagrama se deben considerar los resultados que se pueden tener en la herramienta ya que los nombres que haya en el mismo se reflejarán en la salida generada por ésta. Por esta razón y por posteriores restricciones de implementación, a cada relación de agregación se le va a dar el nombre del elemento al con el que se relaciona, es decir, del elemento contenido. Todas estas relaciones de agregación serán de cardinalidad 0..N debido a que no es preciso tener ningún elemento dentro del diagrama inicialmente y podemos querer tener varios elementos de una clase del modelo en el mismo documento de salida.

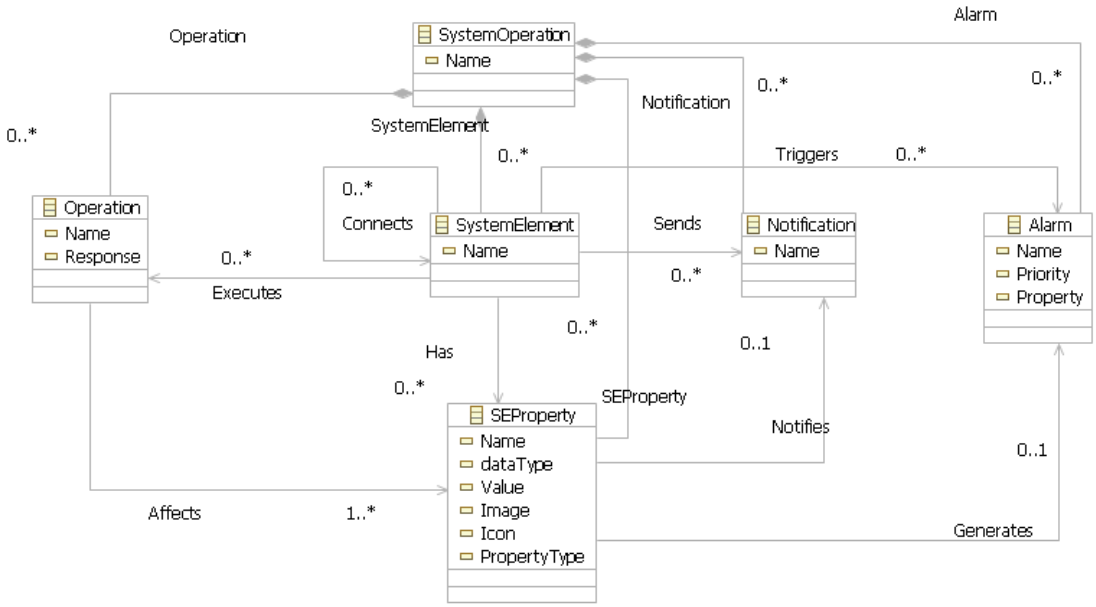


Figura 16. Modelo Ecore definitivo de Espora

Al la hora de utilizar EMF tenemos que tener en cuenta una serie de peculiaridades o reglas. Las relaciones de composición van a suponer un elemento dentro de otro en la aplicación final. Para crear la herramienta conforme a GMF debemos definir un canvas dentro del cual se determinarán las componentes gráficas, es decir, hacemos uso de la relación de inclusión, que en GMF consiste en inclusión gráfica. (ver fig. 17)

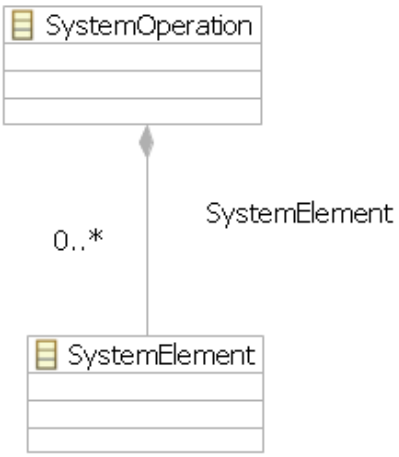


Figura 17. Composición de dos elementos

Para cada asociación debemos determinar una cardinalidad máxima y una mínima, y es singular el hecho de que las cardinalidades de carácter "*a muchos" (cardinalidades N)* se especifiquen con un –1. El rango de cardinalidades abarca desde 0 hasta N ampliando con esto la riqueza de las posibles herramientas a crear.

Para la implementación de cualquier herramienta en GMF necesitamos como punto de partida un tipo de archivo con extensión .ecore, que consiste en un XML que sirve de entrada a GMF para que el programa interprete el modelo que hemos definido, y que ahora en adelante se denominará Ecore. Ecore es el resultado de un diagrama implementado en EMF al cual va asociado Ecore. Aquí almacenaremos todos los elementos con los que se pretenden trabajar en la aplicación, elementos que se puedan "pintar" en el diagrama.

Una vez realizada la composición del metamodelo en el diagrama y su posterior almacenamiento, obtenemos el documento Ecore, el cual posee dos características que se deben especificar, un nombre y una URI. Al asignarle estos identificadores podemos comprobar en la pantalla de acceso a estas propiedades que GMF interpreta gráficamente los componentes definidos en el diagrama del modelo. (Ver Fig. 18)

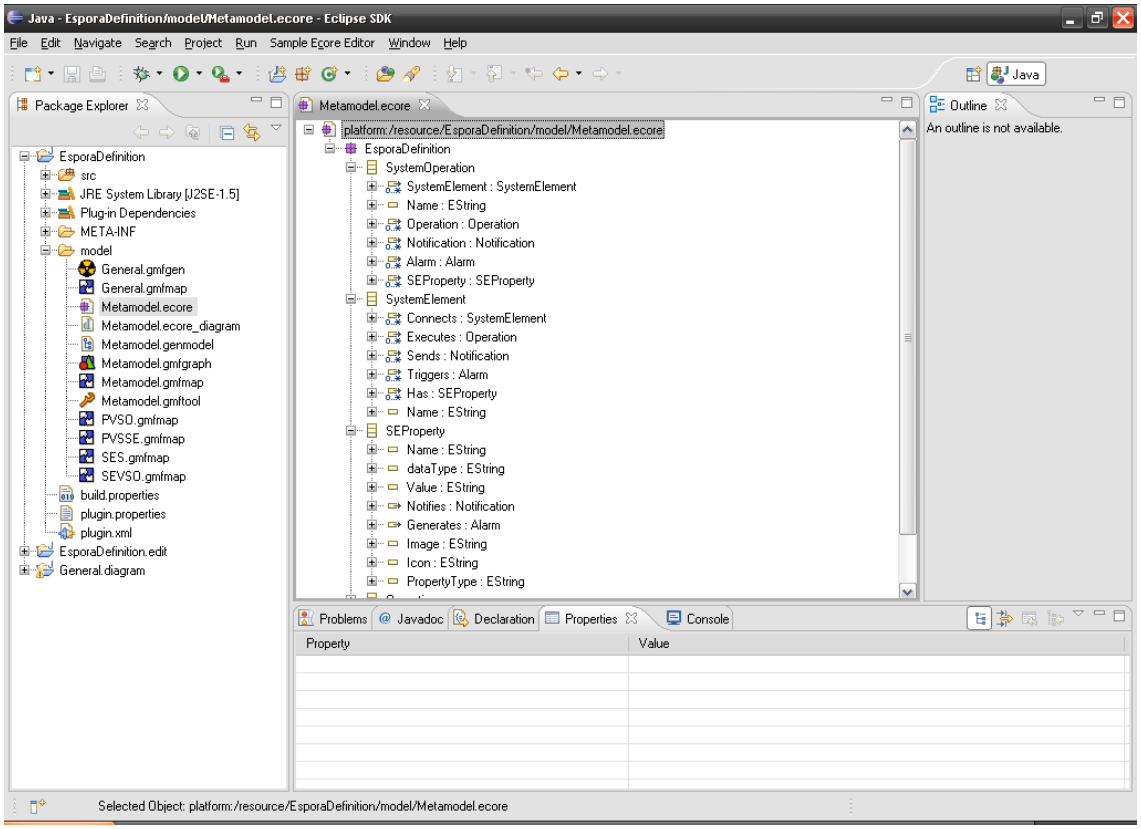


Figura 18. Interpretación gráfica de Ecore por GMF

5.1.2 De Ecore a EMF

Concluida la descripción del modelo, los cambios con respecto al modelo inicial y la configuración que le corresponde al documento Ecore, pasamos a describir el proceso de introducción de Ecore en EMF. En este proceso crearemos un archivo con extensión .genmodel, el cual permitirá la generación automática de código correspondiente a las clases y el editor gráfico del modelo.

Genmodel permite diversas funciones (ver Figura 19), pero entre ellas, las más destacadas son especificar el sitio donde guardar el código generado. Para especificar el sitio en el que debe buscar el generador para realizar su función de forma consistente, se tiene que especificar un paquete o lugar de referencia. Acto seguido generamos el editor y el código de forma

automática clicando con el botón derecho sobre el generador y eligiendo las opciones "*Generate Edit Code*" y "*Generate Model Code*". Así crearemos el código asociado a las clases contenidas en el modelo.

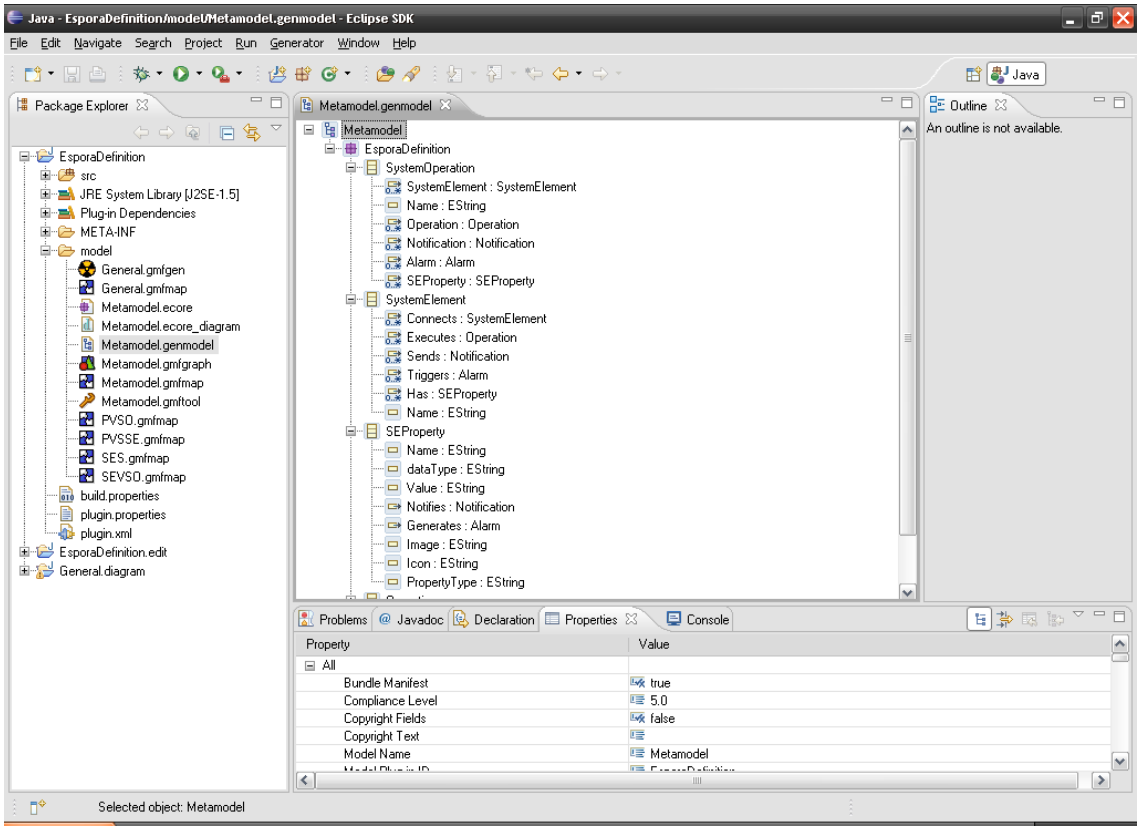


Figura 19. Interpretación de Ecore por GMF (Genmodel)

5.1.3 Definición gráfica

Terminado el proceso de generación automática de código correspondiente a las clases del modelo, proseguimos con el entorno GMF, en particular con la definición de las metáforas gráficas asociadas a los elementos del modelo.

Consideramos que tanto la asociación de ideas de las componentes gráficas como el aspecto meramente estético son importantes para la usabilidad de la herramienta, y por esta razón,

tantos los colores como las formas están firmemente pensados y meditados para una mayor comodidad del usuario.

Debemos saber que cada elemento del modelo corresponderá a un elemento gráfico y cada relación se visualizará con un link gráfico. Dichas correspondencias se representan en la figura 20.

En Espora, **SystemElement** viene representado con una elipse de color azul. Se decidió de esta forma dada la globalidad que implica la figura, y el azul, un color neutro.

La componente **Property** viene representado en forma de rectángulo por las características que encierra, simulando un componente de añadido y de forma sólida. El color que le caracteriza es el verde.

La componente **Operation** viene representado con un rectángulo, pero esta vez con las esquinas suavizadas y un marco rayado, simulando que no supone ni la globalidad de *SystemElement* ni la solidez de *Property*. El color elegido es el amarillo.

La componente **Alarm** viene representado por un rectángulo con las esquinas suavizadas, similar al componente *Operation*, pero de color rojo.

La componente **Notification** es similar a la componente *Alarm* y la componente *Operation*, pero de color naranja.

En cuanto a los enlaces entre elementos, se ha tratado de que el color del elemento al que llega el enlace sea el mismo que el del enlace. Así, tenemos la relación **Has** entre *SystemElement* y *Property* de color verde con línea sólida, la relación **Affects** entre *Operation* y *Property* de color verde con línea sólida, la relación **Executes** entre *SystemElement* y *Operation* de color amarillo con líneas sólidas, la relación **Notify** entre *SystemElement* y *Notification* de

color naranja con línea sólida, la relación **Triggers** entre *SystemElement* y *Alarm*, roja y con líneas sólidas, la relación **Sends** entre *Property* y *Notification*, naranja y con líneas punteadas, la relación **Generates**, entre *Property* y *Alarm*, roja y con líneas punteadas y por último la relación **Connects**, negra, con líneas punteadas y más ancha que las demás debido a su carácter de enlace entre dos *SystemElement*.

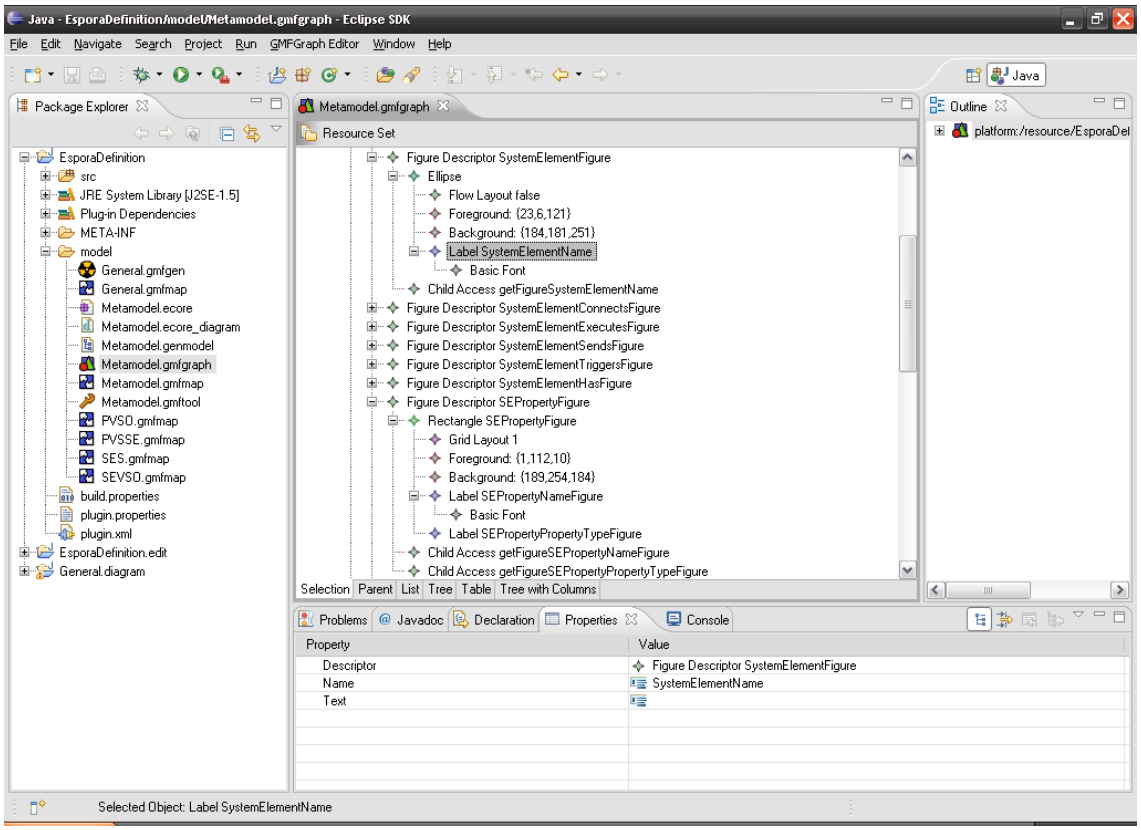
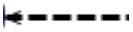


Figura 20 Extracto de la definición gráfica de Espora

El nombre que se le da a cada elemento así como sus características se denota con una etiqueta en negrita (Ver SystemElement, Fig. 15). Así mismo se hizo con el resto de propiedades, pero en letra con formato normal (Ver SEPorperty Fig. 15). En caso de ser más de una característica se decidió ponerlas en formato de columna, por cuestiones de espacio y evitar el recargado del entorno de trabajo. Los nombres de los elementos, con el fin de distinguirlos del resto de características, se implementaron en negrita.

	SEProperty
	Operation
	Notification
	Alarm
	SystemElement
	SEPorperty Generates Alarm
	Operation Affects SEProperty
	SEProperty Notifies Notification
	SystemElement Has Propety
	SystemElement Connects SystemElement
	SystemElement Sends Notification
	SystemElement Executes Operation

	SystemElement Triggers Alarm

Fig. 16 Metáforas gráficas.

5.1.4 Definición de la paleta de herramientas

Esta fase corresponde a la implementación de la paleta de herramientas que va a ofrecer la aplicación, por lo tanto se deben definir tanto las primitivas de modelado que queramos visibles como los iconos que deseemos que aparezcan asociados a las mismas. Por esa razón, el proceso de selección de los iconos ha sido arduo y meticuloso, combinando distintos colores y formas para conseguir el resultado deseado.

A cada primitiva hay que asignarle un icono grande y uno pequeño. Esto ocurre porque GMF facilita un acceso directo a la paleta al posicionarse sobre el Canvas. Los iconos que se han elegido son los mismos para ambos casos, por evitar confusiones del usuario.

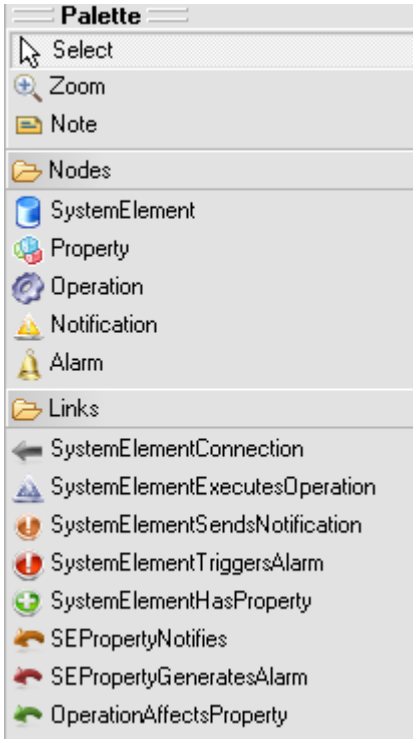


Figura 21. Paleta Espora

El aspecto que se deseaba era un diseño impactante, que a la vez fuese funcional, accesible y sobre todo usable para cualquier tipo de usuario, por eso, con los iconos se trata de llegar a un modelo intuitivo basado en las formas cotidianas que existen a diario.(Ver Fig. 17) Los colores de los iconos tratan de encajar en color con la metáfora asociada a cada elemento del modelo lo que supone un mayor nivel de intuición, ya que se asocian tanto formas como colores y sube el nivel de abstracción por parte del usuario. (Ver Fig. 22)

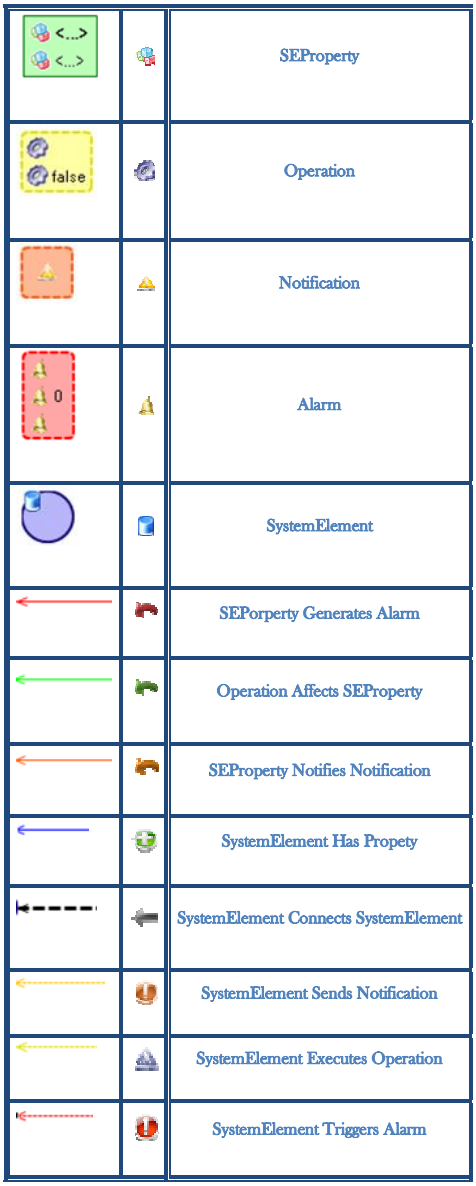


Figura 22. Iconos y metáforas gráficas Espora

5.1.5 Correspondencia Herramientas-Primitivas de modelado-Definición Gráfica

El paso de correspondencia es uno de los más importantes en GMF, ya que es el proceso por medio del cual se asocian la definición gráfica con sus respectivas primitivas de modelado. En definitiva, es la pieza que hace que todo encaje. A esto se debe añadir la posibilidad de realizar "vistas". Llamamos vistas al concepto de poder seleccionar qué elementos del modelo que estamos implementando queremos que se muestren en cada ocasión. En Espora se definen cinco vistas bien diferenciadas:

- La primera de ellas es *SystemElement* y su relación *Connects*.
- La segunda es *SystemElement* con *Property* y su relación *Has*.
- La tercera es *Property* y las operaciones(*Operation*, *Alarm* y *Notification*) junto con sus relaciones.
- La cuarta es *SystemElement* con operaciones(*Operation*, *Alarm* y *Notification*)
- La quinta y última es una vista general en la que se aprecian todos los elementos.

Para realizar el paso de correspondencia se deben elegir qué elementos se desea que aparezcan en la aplicación final de todas las primitivas que existen en el modelo(Ver Fig. 19). Cada una de las correspondencias que hagamos se llamará vista, ya que son las posibles formas de ver el modelo que se cree. Para esto debemos generar correspondencias tantas como vistas queramos especificando en cada correspondencia los elementos que se quieren mostrar en cada vista. (Ver Fig. 23)

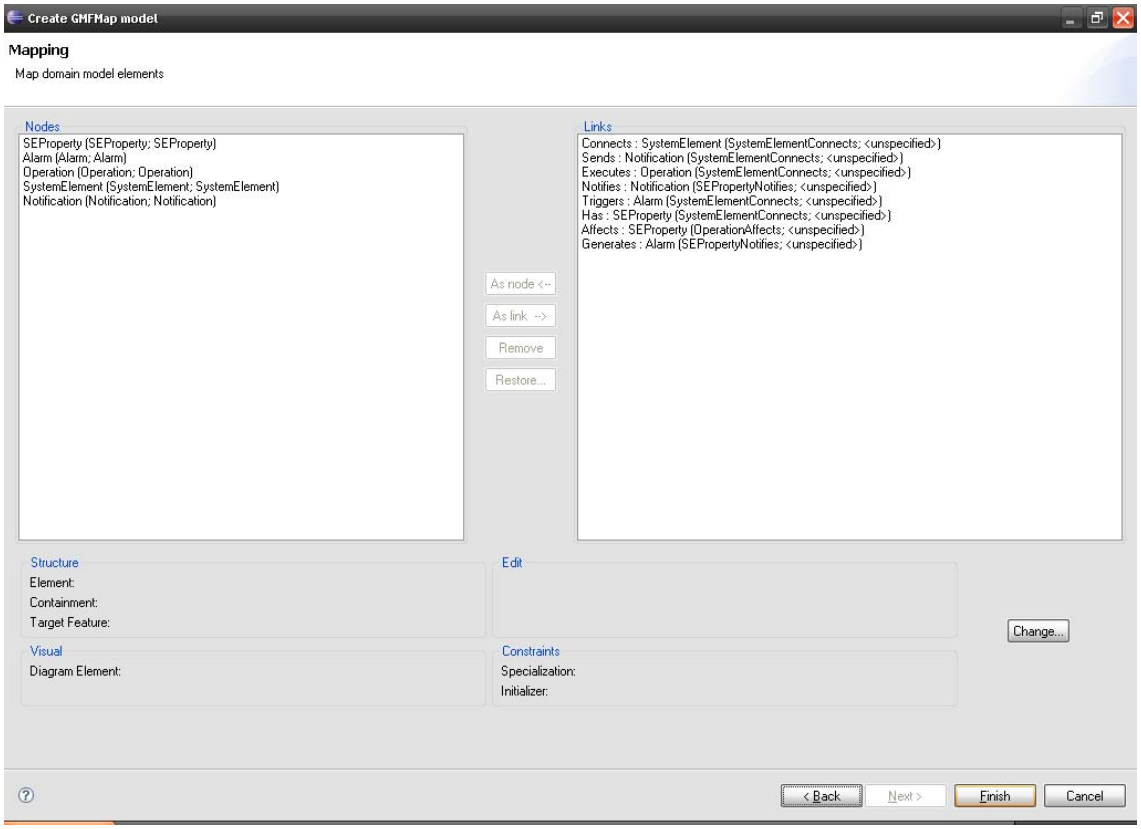


Figura 23. Correspondencia de elementos

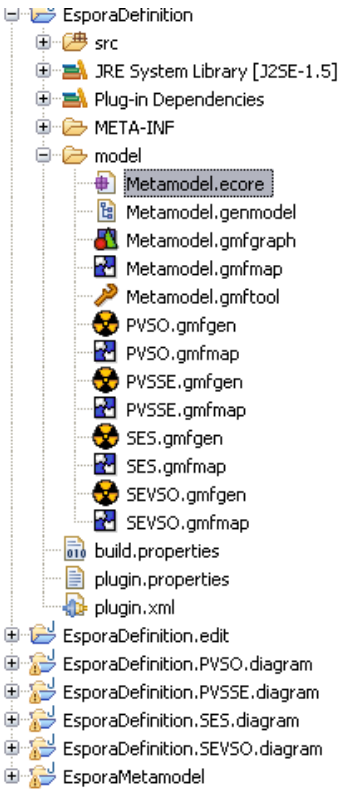


Figura 24. Correspondencias de Espora

5.1.6 Especificación de los Generadores.

De las correspondencias lo que nos interesa verdaderamente es el hecho de que pueden crear generadores de vistas, que va a ser lo que ofrezca nuestra aplicación. Con respecto a esto, de cada mapping se debe crear un generador, y por tanto una vista, especificando en las opciones del generador distinta extensión de diagrama y de archivo xml asociado a esta vista, así como la creación del diagrama en un paquete distinto del que viene por defecto. Cambiaremos también el nombre tipo del diagrama. Los nombres de las opciones se han pensado con siglas, es decir, si la vista corresponde a SystemElement y operaciones el nombre por siglas será SEvsO(SystemElement VS Operations). Una vez especificadas las opciones de los generadores pasamos a generar automáticamente el diagrama, y se nos creará un paquete que contendrá el diagrama con las herramientas y la definición gráfica especificada en el mapping. Cuando se ejecute la aplicación, el menú contextual nos ofrecerá distintas opciones a elegir entre los diagramas que habremos generado con los nombres especificados en el gmfgen.

6. Caso de estudio

Una de las principales ventajas que tiene trabajar con herramientas independientes de dominio es que al no tener un dominio en el que tengan que apoyarse para el desarrollo proporcionan una versatilidad bastante amplia ya que su uso se puede extrapolar a múltiples dominios. En este capítulo se incluye un caso de estudio que permite comprobar la utilidad de la herramienta creada. Dicho caso de estudio consiste en la definición del lenguaje de operaciones para un dominio concreto, una planta genérica de producción de biogás, mediante la herramienta software creada en este trabajo, ESPORA.

5.1 Descripción del caso de estudio

Los objetivos de una planta de biogás son por un lado, generar energía eléctrica a partir del biogás producido en la planta y por otro lado, obtener bio-fertilizantes. Nos centramos en planta que producen biogás a partir de residuos cárnicos. Consta de varias etapas: trituración, pasteurización, homogenización y digestión anaeróbica. En la figura 25 se muestra una torre de las etapas realizadas para obtener el biogás.

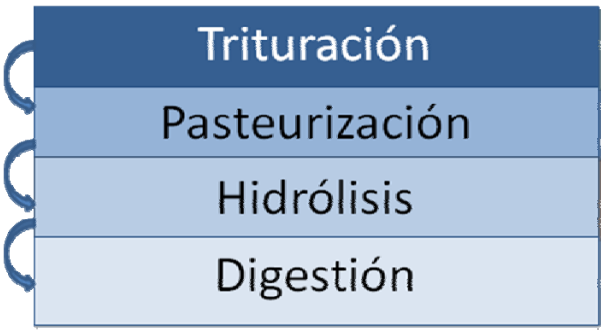


Figura 25. Torre de etapas para la obtención de biogás.

Se define planta de biogás como «*Planta en la que se proceda a la degradación biológica de productos de origen animal en condiciones anaerobias para la producción y recogida de biogás.* »

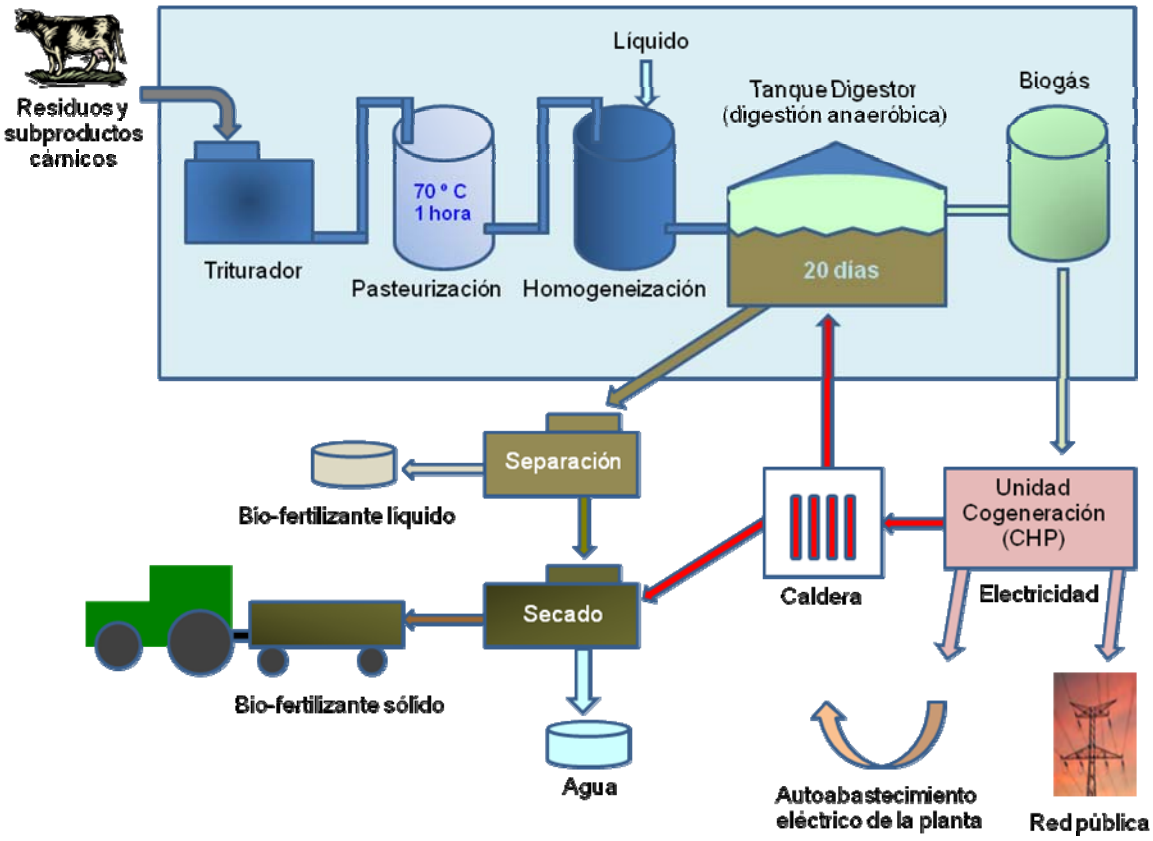


Figura 26. Esquema de funcionamiento de la planta de biogás del cliente

En la figura 26 se muestra de forma esquemática el proceso utilizado por el cliente en su planta para la síntesis de biogás y bio-fertilizantes. Resulta de interés en este caso de estudio la sección del proceso incluida en el recuadro superior. En este recuadro se incluyen los siguientes procesos o etapas:

1. **Etapa de trituración**

   En esta etapa se lleva a cabo la trituración de restos orgánicos sólidos procedentes de:

   - Subproductos animales

- Contenidos gastrointestinales: intestinos porcinos, restos de vacuno, etc.
- Fangos de depuradoras procedentes de industrias agroalimentarias
- Residuos lácteos

Estos restos se introducen en una trituradora (manual o automática) y por medio de un tornillo sin fin se conducen al interior del tanque de pasteurización.

2. **Etapa de Pasteurización**

   Los restos procedentes de la etapa anterior se tratan térmicamente a unos 70º C con el fin de esterilizar la materia, durante un tiempo no inferior a 1 hora. Además, se introduce agua para hacer una mezcla más homogénea. Realizada la pasteurización la masa se introduce en el tanque de homogeneización.

3. **Etapa de Homogeneización**

   En el tanque de homogeneización se tritura nuevamente la biomasa y se mezcla con agua para hacer la mezcla más homogénea. La materia ha de permanecer en el tanque de hidrólisis al menos durante 1 hora. Después se introduce la biomasa en el tanque digestor.

4. **Etapa de Digestión**

   Esta etapa es la más importante del proceso, dentro del tanque digestor la biomasa se trata de forma anaeróbica durante un periodo de 20 días, obteniéndose biogás y lodos orgánicos de los que se obtiene bio-fertilizante.

Para que el proceso funcione correctamente se deben de controlar una serie de factores como el pH, temperatura, nutrientes y tiempo de resistencia, lo que hemos llamado variables de entorno (ver capítulo 4).

5.2 Modelo de operaciones

Todas y cada una de las etapas que forman parte del proceso de biogás, trituración, pasteurización, homogenización y digestión anaeróbica, se realizan en una serie de tanques conectados entre sí para realizar un proceso secuencial, tanque de trituración, tanque de pasteurización, tanque de digestión y tanque de hidrólisis respectivamente. Cada uno de los tanques tiene una serie de variables que el sistema debe ser capaz de monitorizar, controlar o monitorizar y controlar. A continuación se exponen en detalle:

- El **tanque de trituración (ver Figura 27)** : su objetivo es triturar productos cárnicos. Está formado por un triturador y unas compuertas (compuerta de entrada y compuerta de salida del tanque). Tiene además un mezclador encargado de mezclar los productos cárnicos con agua.
    - Modo de puesta en funcionamiento:
        - Tanque trituración: arranque o parada
        - Una vez arrancado el tanque ya podemos poner en funcionamiento sus componentes.
    - Apagado del tanque:
        - Tanque trituración: el apagado del tanque de trituración consiste en poner su estado a OFF. Si alguno de los componentes que lo forman está encendido se debe proceder a su apagado (apagado triturador y, por tanto, su velocidad será 0, y cierre de las compuertas)

Si se produce algún error en el apagado se enviará un código de error para indicarlo.

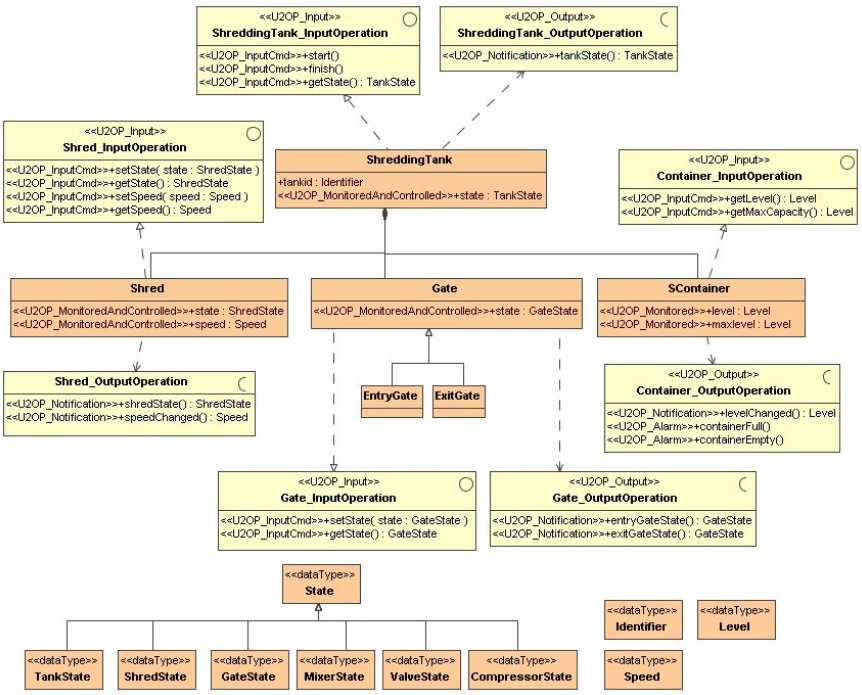


Figura 27. Modelo de operaciones del tanque de trituración

- El **tanque de pasteurización**: su objetivo es esterilizar productos cárnicos. Está formado por un mezclador y unas resistencias de calentamiento. El mezclador mezcla productos cárnicos con agua y las resistencias calientan la mezcla a unos 70º, a esta temperatura se la denomina temperatura nominal.
    - Puesta en funcionamiento del tanque:
        - Tanque pasteurización: arranque o parada
        - Una vez arrancado el tanque ya podemos poner en funcionamiento sus dos componentes. Estos componentes pueden arrancar y parar en modo manual o automático:

- Mezclador
  - Arranque automático: Antes de proceder al vaciado del pasteurizador en modo automático.
  - Parada automática: Al quedar vacío el depósito.
- Resistencias
  - Arranque automático: la temperatura del depósito está más de 22 segundos por debajo de la temperatura de histéresis.
  - Parada automática: el depósito se queda durante 5 segundos por debajo del nivel mínimo o cuando la temperatura del depósito está durante 22 segundos por encima de la temperatura nominal.

El resultado al final del proceso en este tanque es la **biomasa.**

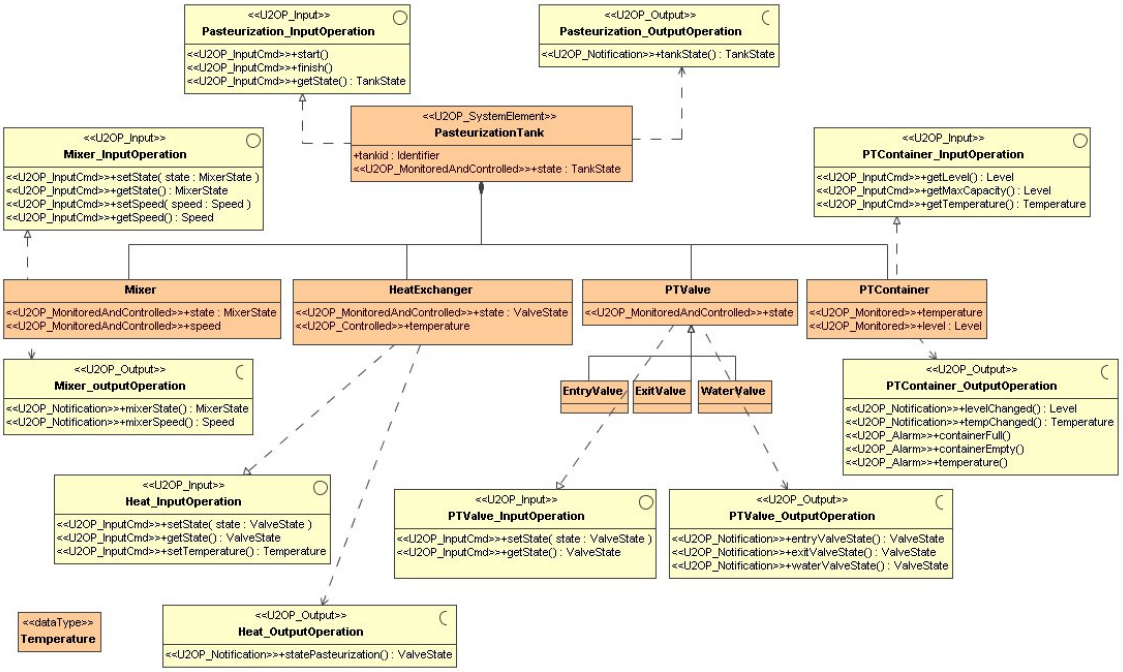


Figura 28. Modelo de operaciones del tanque de pasteurización

- El **tanque de hidrólisis (ver Figura 29)**: su objetivo es mezclar la biomasa con agua. A esta etapa se le denomina como etapa de homogenización. Para la correcta realización de este proceso la materia ha de permanecer en el tanque durante al menos una hora. El tanque está formado por un triturador, una bomba de carga, un mezclador, una válvula de agua caliente y una válvula de paso de la biomasa.
    - **La bomba de carga** transporta biomasa del tanque pasteurizador al triturador y viceversa, y funcionar en modo manual o automático, aunque la descarga sólo se puede hacer en modo automático.
        - Carga automática: 1 minuto después de que se active la variable DB4.DBX.310.0.
        - Descarga automática: Cuando el nivel del depósito sea el nivel de llenado o parada de calentamiento del tanque pasteurizador.

    -**El triturador** tritura biomasa procedente de la bomba de carga. Puede funcionar en modo manual o automático
        - Arranque o parada automática: cuando arranca o para la bomba de carga.

    -**El mezclador** mezcla la biomasa triturada procedente del triturador con agua. Puede funcionar en modo manual o automático.
        - Modo automático:
            - Arranque automático: Siempre que está arrancada la bomba de carga.
            - Comportamiento intermitente controlado por las variables tiempoON y tiempoOFF.

- Parada automática: cuando el nivel del depósito de hidrólisis es inferior al 30%.

-**La válvula de agua caliente** calienta la biomasa del tanque de hidrólisis. Puede funcionar en modo manual o automático.

- Apertura automática: tras 20 segundos consecutivos en los que la temperatura del depósito esté por debajo de la temperatura nominal.
- Cierre automático: tras 20 segundos consecutivos en los que la temperatura del depósito esté por encima de la temperatura nominal.

-**La válvula de paso de biomasa:** pasa biomasa entre tanque triturador y el mezclador. La apertura automática se produce al arrancar la bomba de carga del digestor y el cierre automático se produce 4 segundos después de parar la carga de la bomba.

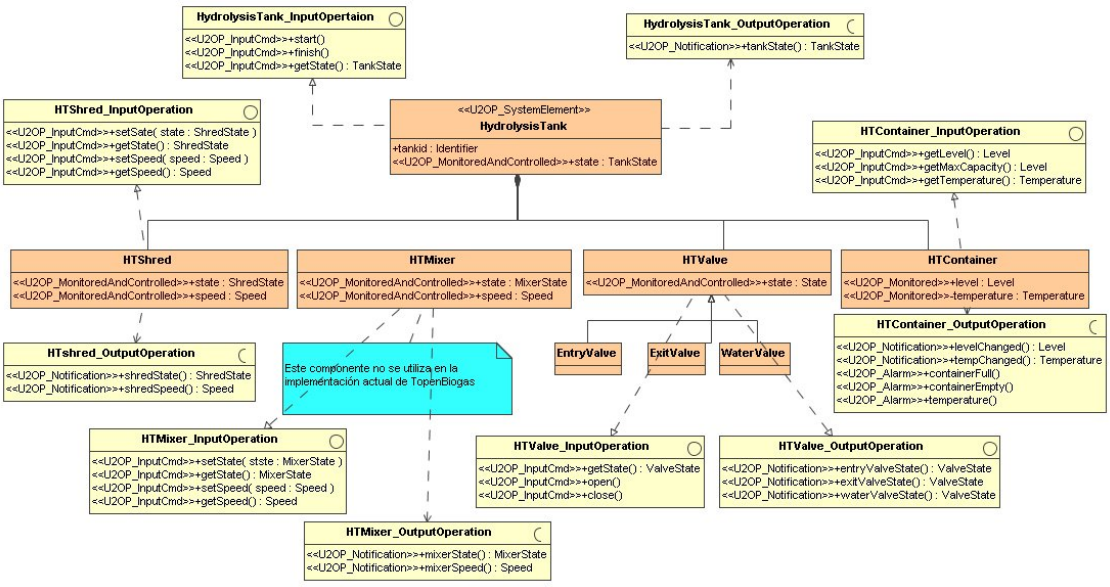


Figura 29: Modelo de operaciones del tanque de hidrólisis

- El **tanque digestor:** su objetivo es el tratamiento de la biomasa de forma anaeróbica para obtener biogás y residuos orgánicos. Para un resultado óptimo en el proceso, la materia debe permanecer en este tanque 20 días. Está formado por un compresor de 8 bar, una bomba de carga, una válvula de paso de biomasa, un digestor, una válvula de agua caliente, un compresor de membrana y una bomba de extracción.

  -La **válvula de paso de biomasa** da paso a la biomasa desde el tanque de hidrólisis al tanque digestor. Puede ser abierta o cerrada de forma manual.

  -El **compresor de 8 bar**: Sirve para dar presión a la bomba de vaciado del tanque de hidrólisis hacia el tanque digestor. Arranca automáticamente al arrancar la bomba de carga. Sólo se puede parar manualmente.

  -La **bomba de carga** transporta biomasa del tanque de hidrólisis al tanque digestor y viceversa. Puede funcionar en modo manual o automático.

  - Arranque automático:
    - Descargar la biomasa que contenga el tanque de hidrólisis hasta que el nivel de llenado del digestor esté por debajo del nivel máximo.
    - Proceso de carga.
  - Parada automática:
    - Cuando ya ha pasado la cantidad de biomasa calculada para la secuencia.
    - Cuando el nivel del depósito del tanque de hidrólisis baja del 30%
    - Si se cierra la válvula de paso de biomasa entre el depósito del tanque de hidrólisis y el tanque digestor
    - Si se para el compresor de 8 bar.

- El **mezclador** puede funcionar en modo manual o automático.
    - Arranque automático:
        - Cuando se active la variable db4.dbx.310.3
    - Parada automática:
        - Cuando se active la variable db4.dbx3.1
        - Cuando se produzca un error en el motor
        - Cuando esté trabajando la carga de biomasa a través de la bomba de carga.
        - Cuando esté trabajando la bomba de vaciado del tanque digestor.

-La **válvula de agua caliente**: calienta la biomasa del tanque digestor. Puede funcionar en modo manual o automático.

- Apertura automática: tras 20 segundos consecutivos en los que la temperatura del depósito esté por debajo de la temperatura nominal (por defecto será 30º).
- Cierre automático: tras 20 segundos consecutivos en los que la temperatura del depósito esté por encima de la temperatura nominal.

-El **compresor de membrana** se encarga de mantener al tanque digestor a la presión adecuada para que se lleve a cabo la digestión anaeróbica.

-La **Bomba de extracción** sólo actúa durante 10 segundos. Puede funcionar en modo manual o automático.

- Apertura automática:
    - Cuando se activa el primer paso de la carga del tanque digestor a través de la bomba de carga.
    - Al inicio de la secuencia automática de carga y descarga del tanque digestor.

- Parada automática: cuando se queda el depósito del tanque digestor sin biomasa.

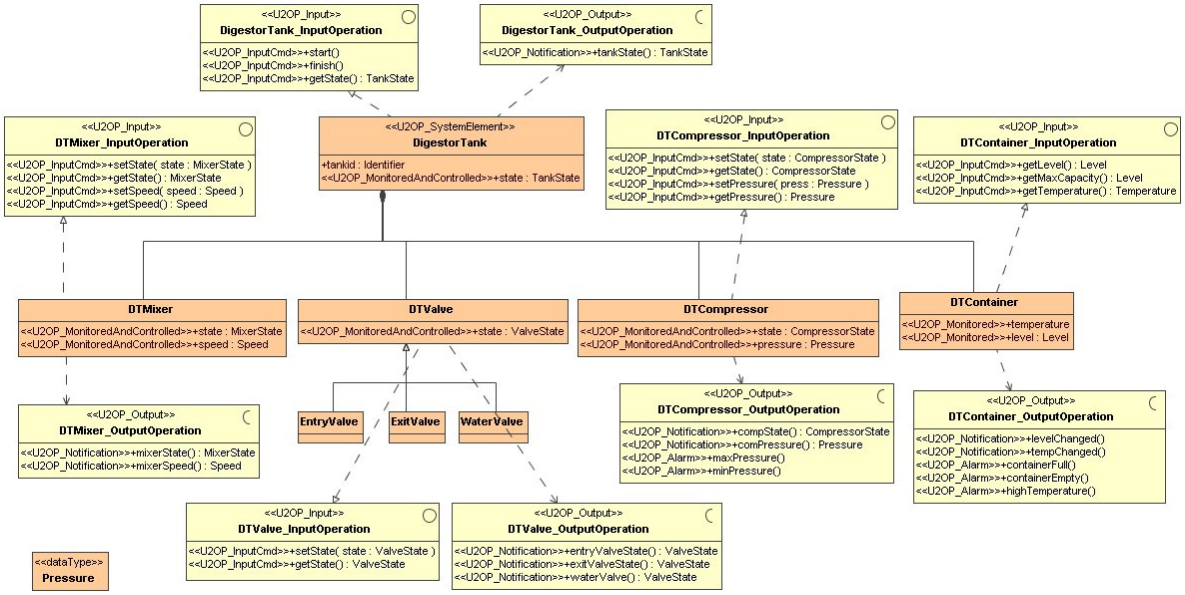


Figura 30. Modelo de operaciones del tanque digestor

6.3 Modelado con ESPORA.

En la figura 26 se ha mostrado que el proceso seguido para obtener el biogás se trata de un proceso secuencial en que hay varios tanques implicados. Como ya se vio en el capítulo 5, cada tanque en el contexto de ESPORA está representado por un SystemElement y cada relación entre tanques por una relación de tipo SystemElementConnection (Ver figura 31).

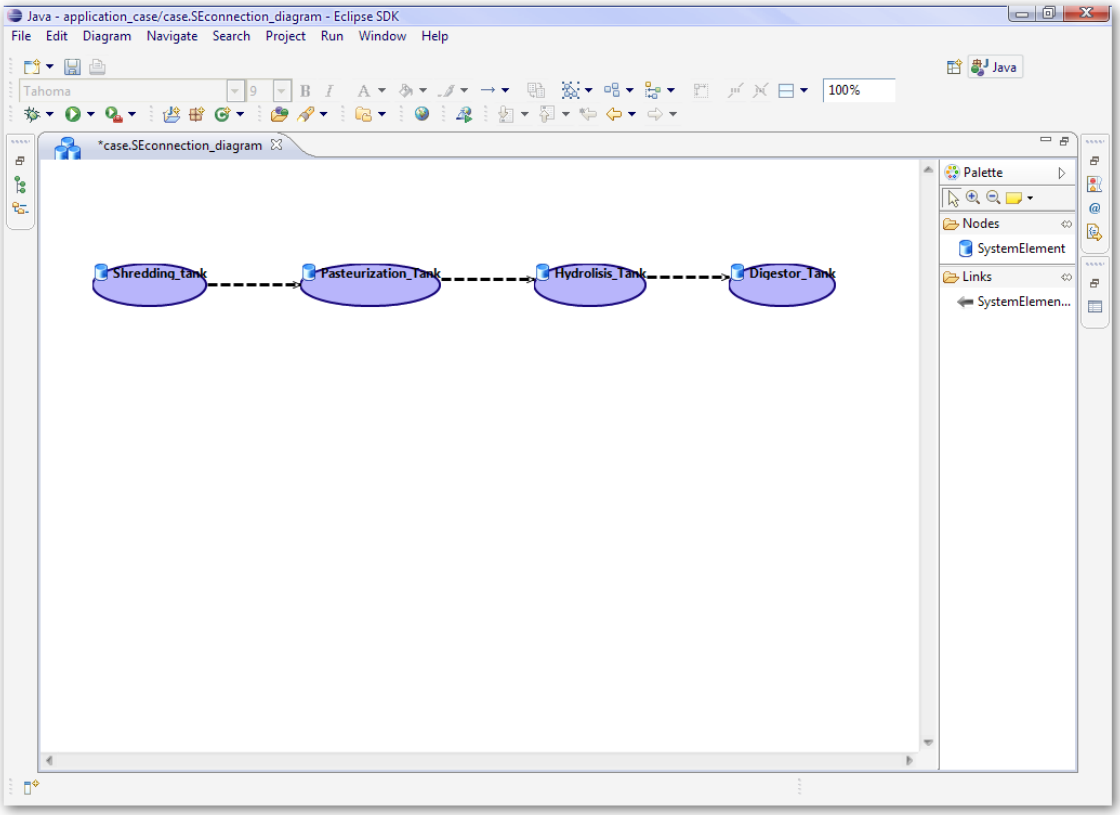


Figura 31. Modelado de la conexión de los tanques.

Debido a que ESPORA tiene 5 vistas diferentes, se permite mostrar de forma individualizada cada conjunto de relaciones haciendo el modelado mucho más sencillo y accesible. En la figura 32 se puede observar una vista de los tanques de biogás con sus respectivas operaciones.

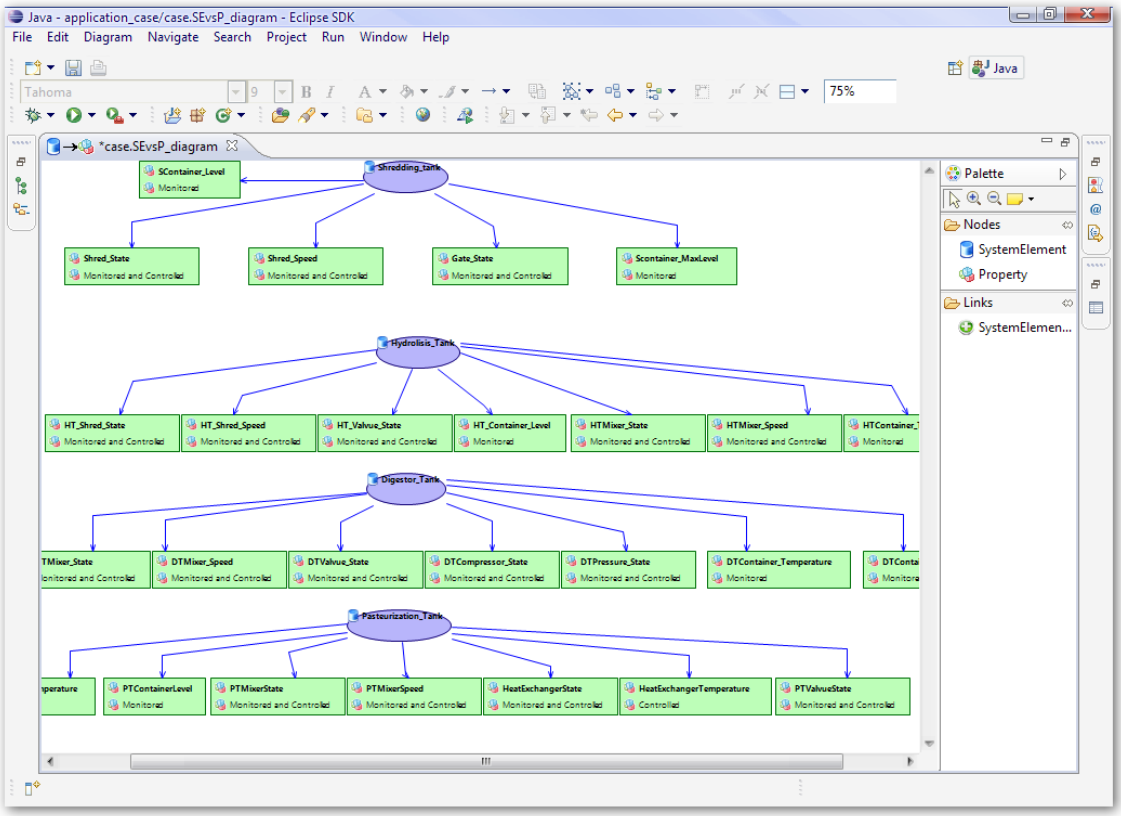


Figura 32. Modelado de los tanques con sus operaciones

Por último, en la figura 33 se muestra una vista parcial de los tanques y sus operaciones de entrada y avisos de salida, en concreto se muestra el tanque de trituración.

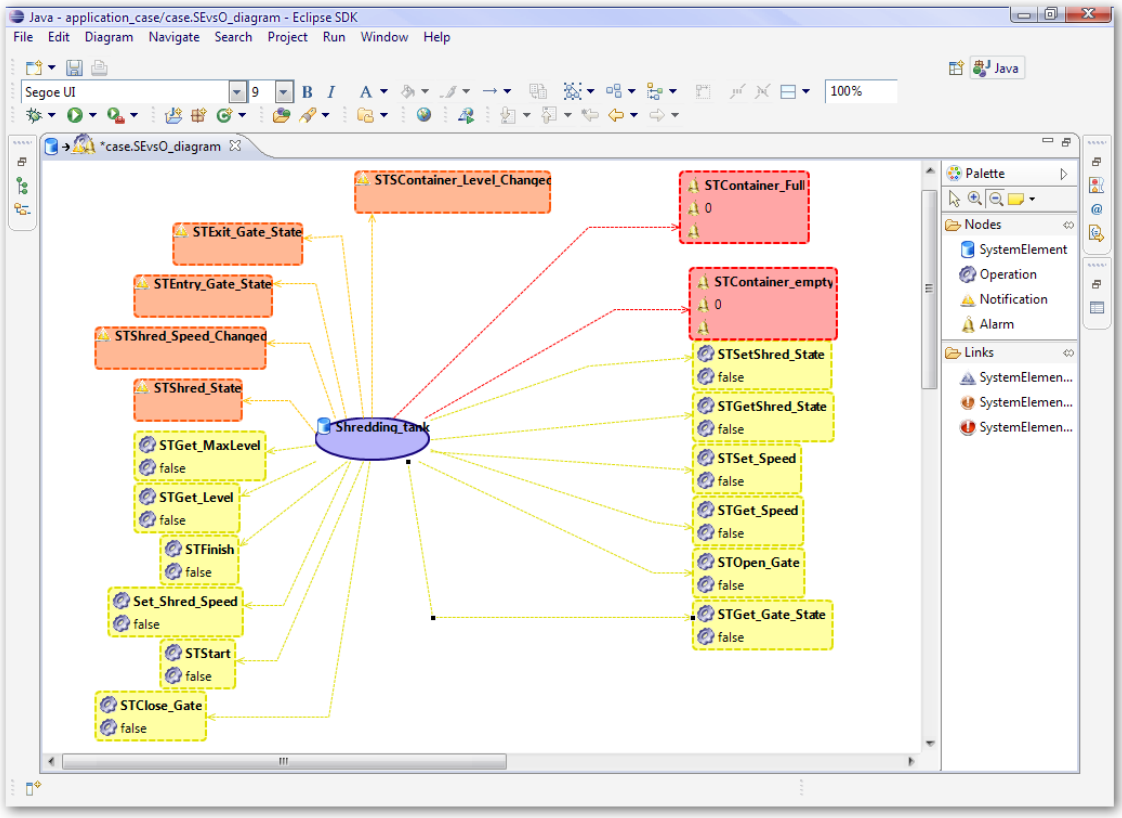


Figura 33: Tanque de trituración con sus operaciones, notificaciones y alarmas.

Bibliografía

[Ala08] Alarcón, P.P.: Especificación de un entorno de operación aplicable a procesos de desarrollo y operación de sistemas con software. PhD thesis, Facultad de Informática. Universidad Politécnica de Madrid (2008).

[Ala07] Alarcón, P.P., Garbajosa, J.: Identifying application key knowledge through system operations modeling. In: Proceedings of the 6th IEEE International Conference on Cognitive Informatics (ICCI'07), Lake Tahoe, California, IEEE CS Press (2007) 246-254

[AMM08] The AMMA (ATLAS Model Management Architecture) platform , http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/

[Amb04] Ambler, S. (2004). Agile Model-driven Development with UML 2.0, Cambridge University Press.

[Bey05] Beydeda, S., Book, M., & Gruhn V. (2005), Model-Driven Software Development, Springer, 2005.

[DSL06] Domain-Specific Language Tools (2006) DSL Tools, http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx

[ECL08] Eclipse. Eclipse - an open development platform. http://www.eclipse.org/

[EMF08] EMF: Eclipse Modeling Framework Project (EMF) http://www.eclipse.org/modeling/emf/

[Fon04] Fondement F., Silaghi R., Defining Model Driven Engineering Processes, Third International Workshop in Software Model Engineering (WiSME), held at the 7th International Conference on the Unified Modeling Language (UML), Lisbon, Portugal, October 11-15, 2004.

[GME08]GME: Generic Modeling Environment, Institute for Software Integrated Systems, Vanderbilt University, Nashville, USA, http://www.isis.vanderbilt.edu/Projects/gme/

[GMF08] GMF: The Eclipse Graphical Modeling Framework (GMF), http://www.eclipse.org/modeling/gmf/

[Gre04] Greenfield J., Short K, Cook S., and Kent S. Software Factories. Wiley Publising Inc., 2004.

[Kle04] Kleppe A., Warmer J., Bast W., MDA EXPLAINED The Model Driven Architecture: Practice and Promise, Addison Wesley, Object Technology Series, Grady Booch, Ivar Jacobson, and James Rumbaugh, 2004.

[Mag08] Magro B., Garbajosa J., Pérez J., A Software Product Line Definition for Validation Environments, The 12th International Software Product Line Conference (SPLC08), IEEE Computer Society, Limerik, Ireland, 8-12 septiembre 2008.

[MDA08] Model Driven Architecture Guide, Object Management Group, http://www.omg.org/docs/omg/03-06-01.pdf

[MED08] Medini QVT. http://projects.ikv.de/qvt

[MOD08] ModelWare: ModelWare European project , http://www.modelware-ist.org/

[MOF08] Meta-Object Facility (MOF) 1.4 Specification., Object Management Group (OMG),TR formal/2002-04-03.http://www.omg.org/technology/documents/formal/mof.htm

[QVT08] Query View Transformation (QVT), Object Management Group (OMG) (2007): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Final Adopted Specification ptc/07-07-07 http://www.omg.org/docs/ptc/07-07-07.pdf

[OAW08] OAW : openArchitectureWare. http://www.openarchitectureware.org/

[Sch06]Schmidt D.C. (2006). Model-Driven Engineering, IEEE computer Society.

[SMA08] Smart QVT(2008). Smart QVT. http://smartqvt.elibel.tm.fr/

[TOG08] Together, Borland Together. http://www.borland.com/

[UML08] The Unified Modeling Language Website, Object Management Group (OMG), http://www.uml.org/

[XML08] Extensible Markup Language (XML), w3C, http://www.w3.org/XML/

[XSL08] XSLT, XSL Transformations (XSLT), http://www.w3.org/TR/xslt